MASTER EN SCIENCES INFORMATIQUES

"Software-based traffic generator"
Leurs performances sur le terrain

Lecomte, Dorian

Award date:
2015

Awarding institution:
Universite de Namur

Université de Namur
Faculté d'informatique
Année académique 2014–2015

"Software-based traffic generator" : Leurs performances sur le terrain

Dorian Lecomte

Promoteur : _____ (Signature pour approbation du dépôt - REE art. 40)
Marie-Ange Remiche

Mémoire présenté en vue de l'obtention du grade de
Master en Sciences Informatiques.

Résumé

Les générateurs de trafic de type "software-based" sont des générateurs de trafic utilisés pour analyser les performances d'architecture particulière en réseau. Des études récentes ont pourtant mis en évidence qu'il existait une différence entre le cahier des charges de ces générateurs et leurs comportements réels. Dans ce travail, on se propose de dégager les problèmes types rencontrés et la source envisageable de ceux-ci.

Avant-propos

Ce mémoire prend naissance à l'issue de mon projet personnel réalisé en troisième année de bachelier. J'avais réalisé ce projet sur les générateurs de trafic IP, de nombreuses questions étaient restées en suspens. Parmi celles-ci, l'envie de comprendre les rouages et mécanismes internes de ces applications, l'envie de comprendre leurs principes fondamentaux. Ils m'intriguaient. J'avais besoin de comprendre.

Une fait attisait particulièrement ma curiosité : les générateurs font partie des applications placées à la croisée de plusieurs champs de recherche. Ainsi, ils entretiennent le dialogue entre les mathématiciens, les ingénieurs et les informaticiens. Les mathématiques cultivent cette notion d'élégance tant décrite par les sciences et techniques. Elles évoluent dans un monde conceptualisé et abstrait. C'est un monde différent de notre quotidien. Les générateurs de trafic permettent une passerelle entre les deux. Ils implémentent ces concepts mathématiques pour mettre en pratique la puissance qui en découle. Or, comme tout modèle scientifique, une implémentation correspond à une abstraction de la réalité. Il n'est pas possible de prendre en compte l'ensemble des paramètres du monde physique. La compréhension plierait sous le poids de la complexité. Dès lors, il existe nécessairement une dissonance entre les modèles et la réalité. L'histoire des sciences en contient nombre d'exemples. Le principe de réinvention perpétuel des sciences est la résultante de l'évolution de ces modèles. Ce processus s'inscrit dans une volonté de toujours obtenir une meilleure adéquation avec notre environnement. En tant que scientifiques, nous devons rester critiques sur les outils que nous utilisons. Nous ne devons jamais oublier les limites de ceux-ci. Il faut pouvoir prendre du recul pour mieux comprendre.

C'est nourri de ces idées que j'ai entrepris la rédaction de ce mémoire. Je souhaitais trouver réponse à des questions d'ordre technique. J'obtins bien plus. Ce fut un véritable enrichissement sur le plan personnel. Jamais je n'aurais pensé voyager autrement que par les livres. Les événements m'ont prouvé le contraire : il fait chaud sous le soleil de Barcelone. Il gravite autour de ce mémoire une formidable expérience humaine. Je me dois de remercier plusieurs personnes sans qui ce mémoire n'aurait pas été possible.

Je voudrais tout d'abord remercier ma promotrice, la professeur Marie-Ange Remiche pour ses judicieux conseils et ses corrections. Ensuite, le professeur Laurent Schumacher pour m'avoir offert l'opportunité de participer au septième *Traffic Monitoring and Analysis Workshop*. J'adresse toute ma reconnaissance aux professeurs ayant répondu à mes questions pendant la rédaction de ce mémoire. À mes amis et à la famille, je vous remercie pour le support moral et intellectuel que vous m'avez apporté tout au long de ce travail. Encore une fois, merci à tous.

Table des matières

Chapitre 1

Introduction

Les scientifiques utilisent des outils pour évaluer les performances de leurs réseaux. Ces outils se basent sur une utilisation du réseau telle qu'on pourrait la décrire au quotidien, ou, au contraire, dans des situations extrêmes. Pour simuler ces utilisations, les scientifiques ne font pas appel aux utilisateurs du réseau à proprement parler mais à des applications capables de générer du trafic. On les appelle *"générateurs de trafic"*. Ce sont des application permettant d'auditionner et de tester les réseaux. Prenons maintenant un générateur de trafic et deux ordinateurs rigoureusement identiques. Nous exécutons le générateur sur chacun d'eux avec les mêmes paramètres. Nous mesurons les flux produits sur le réseau par ces ordinateurs. Un flux correspond au trafic produit par le générateur. Dans le cadre expérimental, intuitivement, nous faisons l'hypothèse que les flux générés par ces deux ordinateurs sont identiques. En effet, il s'agit du même générateur. Or, les mesures indiquent des dissemblances : les débits de ces flux sont différents lorsqu'ils sont comparés.

Dans ce mémoire, nous étudions les dissemblances et l'origine de ces dissemblances. Nous analysons les flux générés par un même générateur de trafic. Nous déterminons s'ils ont un débit statistiquement similaire. Nous identifions quels sont les facteurs extérieurs au générateur capables d'influencer ses performances. Nous nous focalisons exclusivement sur les réseaux IP. La littérature scientifique illustre empiriquement des performances variables pour les générateurs de trafic. Cependant, les études réalisées jusqu'alors adoptent une vision causale. Elles considèrent uniquement le générateur utilisé et le trafic produit en sortie. En conséquence, elles apportent des solutions en modifiant le code source du générateur ou sous forme de recommandations. Elles n'intègrent pas les dépendances du générateur dans le cadre expérimental. Ces dépendances sont intrinsèquement liées aux architectures informatiques modernes. Nous entendons par là l'agencement entre matériel physique, système d'exploitation et application. L'originalité de ce mémoire est d'intégrer ces dépendances dans le cadre de l'étude. Nous réalisons une expérience illustrant l'influence des facteurs externes sur les générateurs. Nous montrons que des changements induits dans le système d'exploitation provoquent une modification des performances d'un générateur de trafic.

Ce mémoire a pris naissance avec la lecture d'un article intitulé *"Do you trust your software-based traffic generator ?"*. Il fut publié par Alessio Botta, Antonio Pescape et Alberto Dainotti de l'université Frédérico II de Naples (voir [19]). Cet article présente une étude comparative des générateurs de trafic. Il montre la grande disparité dans les performances de ces applications.

Nous avons souhaité étudier ce phénomène sous un angle différent au travers de cet ouvrage.Ce mémoire se trouve à la croisée de trois domaines scientifiques : les interfaces homme-machines, les sciences expérimentales et les réseaux informatiques. Pour cela, nous avons utilisé les informations disponibles dans la littérature scientifique, les brochures spécialisées et les livres techniques.

Ce document est composé de sept chapitres. Le premier est la présente introduction. Les chapitres 2, 3 et 4 introduisent les concepts nécessaires à l'expérience présentée au chapitre 5. Ainsi, le second chapitre est une revue de la littérature. Nous y définissons formellement le concept de générateur. Nous y exposons une vue panoramique des applications actuelles. L'originalité de ce chapitre est de présenter l'ensemble des applications en les regroupant par catégorie. À chaque catégorie correspond un scénario d'utilisation type tel que la mesure du débit maximal, la simulation d'un trafic réseau particulier, etc. Nous présentons les générateur de trafic suivant une approche descendante. Nous partons d'un tableau de synthèse représentant l'ensemble des générateurs pour expliquer nos catégories. Ensuite, partant de cette vision d'ensemble, nous sélectionnons un générateur de trafic pour notre expérience et nous justifions notre choix. Le chapitre 3 présente l'architecture générale et les concepts principaux des systèmes d'exploitation. Il présente le système d'exploitation retenu pour notre expérience et la justification de ce choix. Le chapitre 4 présente les principes généraux des réseaux informatiques. Il est spécifiquement orienté sur les technologies matérielles. Ce chapitre présente le réseau retenu pour notre expérience et la justification de ce choix. Le chapitre 5 décrit l'ensemble de notre expérience. Nous exposons d'abord l'ensemble des informations nécessaires à la compréhension et à la reproductibilité de l'expérience. Ensuite, nous présentons les résultats. Le chapitre est structuré sous la forme d'un rapport de laboratoire. C'est la méthode enseignée à la faculté des sciences de l'université de Namur pour la présentation de données expérimentales. Le chapitre 6 expose notre proposition de solution. Nous y décrivons l'architecture d'une suite logicielle intégrant une nouvelle approche pour la conception d'un générateur de trafic. Nous proposons de fusionner le système d'exploitation et le générateur de trafic. Enfin, le chapitre 7 est la conclusion de ce mémoire.

Chapitre 2

Les générateurs de trafic

L'objectif de ce chapitre est de sélectionner un générateur de trafic pour notre expérience. Nous choisirons le générateur parmi ceux rencontrés dans notre revue de la littérature scientifique. Nous procéderons en trois étapes. Premièrement, nous présenterons les générateurs de trafic logiciel à la section 2.1. Nous donnerons une définition précise de ces outils. Ensuite, nous exposerons une vue panoramique de ces applications. Il en existe différentes catégories pour différents usages. Les sous-sections 2.1.2 page 6 et 2.1.3 page 10 présentent respectivement les catégories et la structure de ces catégories. Deuxièmement, nous présenterons le générateur retenu pour notre expérience à la section 2.2 page 16. Nous y justifierons notre choix. Troisièmement, nous présenterons le cahier des charges du générateur retenu à la section 2.3 page 17. Il est composé de deux parties. D'une part, une présentation générale du générateur à la sous-section 2.3.1 page 18. D'autre part, un ensemble de résultats expérimentaux sur les performances de l'application. Nous présenterons la notion de performance et ces résultats expérimentaux à la sous-section F page 115. Enfin, nous terminerons par une discussion sur les informations données dans le cahier des charges à la section 5.6 page 70.

2.1 Présentation des générateurs de trafic

Nous avons donné une définition intuitive des générateurs de trafic dans l'introduction. Nous donnons maintenant la définition complète. Précisons d'abord le vocabulaire. Nous avons besoin des concepts d'application, de logiciel et de programme.

Application :
Une application est un ensemble cohérent d'un ou plusieurs logiciels nécessaires pour exécuter une tâche donnée. C'est un outil destiné à l'utilisateur final. Une application doit pouvoir être utilisée sans connaissances techniques. Par exemple, le montage vidéo et photo requiert des applications telles que Photoshop® ou Sony Vegas Pro®. Une application peut être assistée matériellement. C'est notamment le cas des scanners et des logiciels de mixage audio. Enfin, elle peut devoir être exécutée par un ou plusieurs systèmes informatiques simultanément.

Logiciel :
Un logiciel est un ensemble de programmes, de procédés, de règles et de la documentation relatifs au fonctionnement d'un ensemble de traitements de données.

Programme :

> Un programme est composé de deux éléments. D'une part, une suite d'opérations nécessaires et suffisantes à accomplir pour obtenir un résultat précis. D'autre part, un ensemble de données enregistrées sur un support. Les instructions et les données d'un programme sont susceptibles d'être traitées par un système informatique.

Ainsi, un générateur de trafic se définit comme une application qui rassemble des logiciels, des programmes et des données nécessaires à la génération de trafic sur un réseau informatique. Un réseau informatique se définit comme un ensemble d'entités matérielles et logicielles autonomes interconnectés les unes aux autres par des voies de communication. Le trafic réseau correspond à la circulation et la fréquence des données sur une ou plusieurs de ces voies de communication.

Nous avons identifié dans [50] un des premiers modèles de générateur logiciel. Il s'agit d'un brevet publié en 1995 par la société "Cadence Design Systems inc". Il décrit la conception et l'implémentation d'une méthode de modélisation du trafic basée sur le paradigme client-serveur. Selon Georges et Olivier Gardarin (voir [45]), l'architecture client/serveur se définit comme *"Un modèle d'architecture applicative où les programmes sont répartis entre processus client et serveur communiquant par des requêtes avec réponses"*.

La plupart des générateurs logiciels ont été développés par des équipes de recherche. Pour la majorité, ces applications sont disponibles en accès libre sur internet. Elles sont accompagnées de leur documentation et de conseils pour la mise en oeuvre. Il est également possible d'obtenir le code source. Cela permet d'adapter les générateurs logiciels existants aux besoins de chaque groupe de recherche. Cependant, les scientifiques n'ont pas attendu un accord commun pour développer des générateurs de trafic logiciels. L'absence de norme à des conséquences sur les fonctionnalités et les performances de ces applications. Chaque projet de recherche possède ses propres conventions. Les outils qu'ils construisent pour leurs études en sont imprégnés. Chaque générateur est donc unique et il en existe autant qu'il y a de projets de recherche. En conséquence, il n'est pas possible de considérer toutes ces applications individuellement. Il en existe un trop grand nombre. C'est pourquoi nous avons sélectionné un échantillon. Cet échantillon est composé des vingt-sept générateurs les plus cités dans notre revue de la littérature scientifique. Nous avons pris ceux dont le nombre de citation sur l'outil "Google Scholar" est le plus élevé. Le tableau de la figure 2.1 page 5 reprend ces générateurs. Une vue éclatée de ce tableau répartie sur trois pages est disponible en annexe B de la page 106 à la page 108. Nous justifions l'emploi de ce tableau par notre approche pédagogique : il facilite la compréhension et permet d'étudier des ensembles de générateurs aux caractéristiques communes. Cette méthode est plus adéquate que la présentation d'un ensemble d'outils sans liens apparents entre eux. La sous-section 2.1.1 donne les informations nécessaires à la compréhension de ce tableau. Ensuite, l'analyse de ces générateurs démontre l'existence de cinq profils d'utilisation. Ils correspondent à nos catégories. La sous-section 2.1.2 explique ces catégories. De plus, ces générateurs sont capables de produire différents type de trafic avec des performances variables. Ce constat est explicité à la section 2.1.3. L'annexe A page 91 donne une description pour chacun des vingt-sept générateurs présentés dans ce tableau.

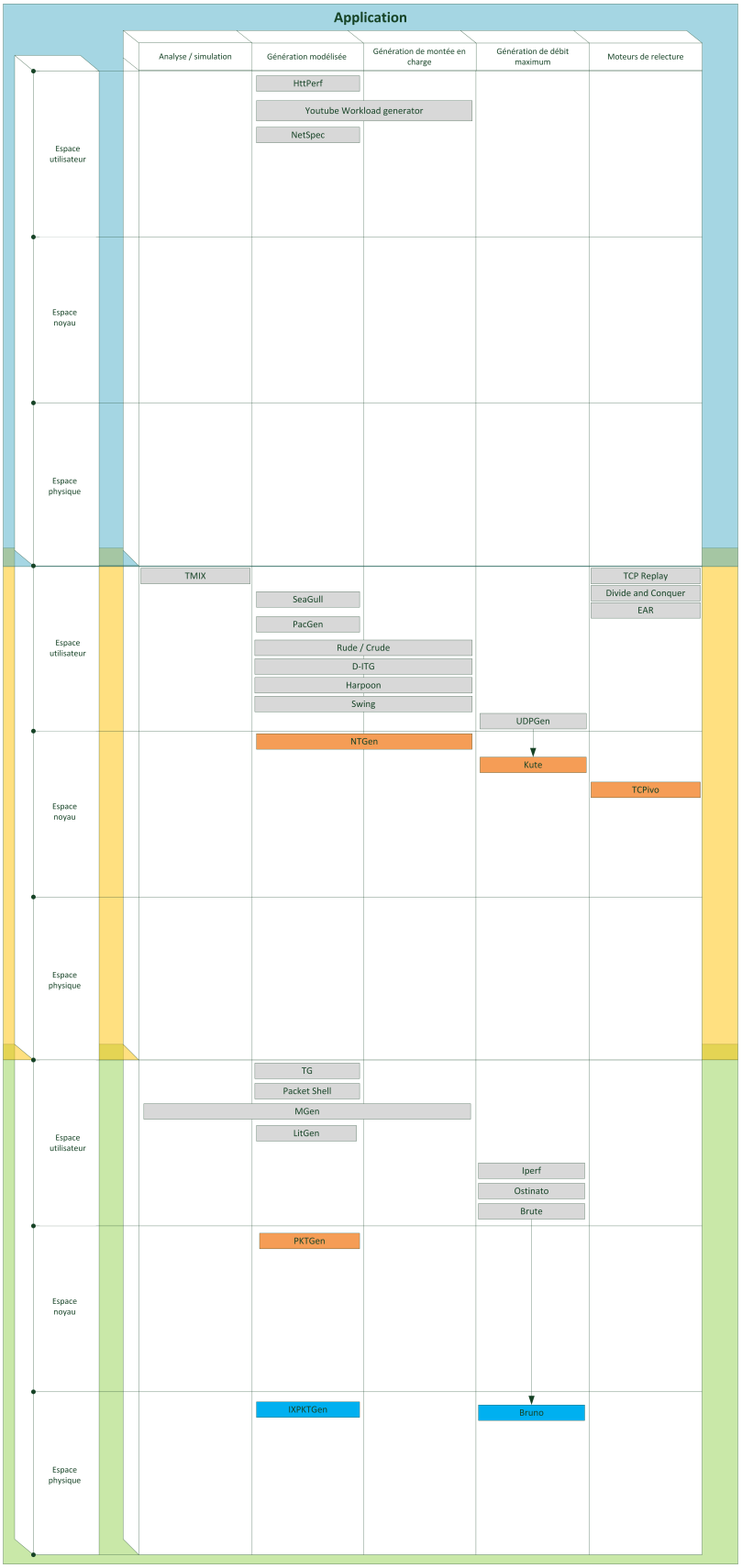

FIGURE 2.1 – Vue d'ensemble des générateurs logiciels

2.1.1 présentation du tableau

Le tableau est à voir comme un building de logement. Ce bâtiment contient cinq appartements identiques de trois étages chacun. Chaque étage compte exactement trois locaux. À nos appartements correspondent des catégories. Elles ont toutes la même structure : trois niveaux de trois espaces. Chaque catégorie contient un type de générateur qui lui est propre. Le schéma de la figure 2.2 illustre cette hiérarchie. Nous présentons les catégories. Ensuite, nous expliquons la structure de ces catégories.

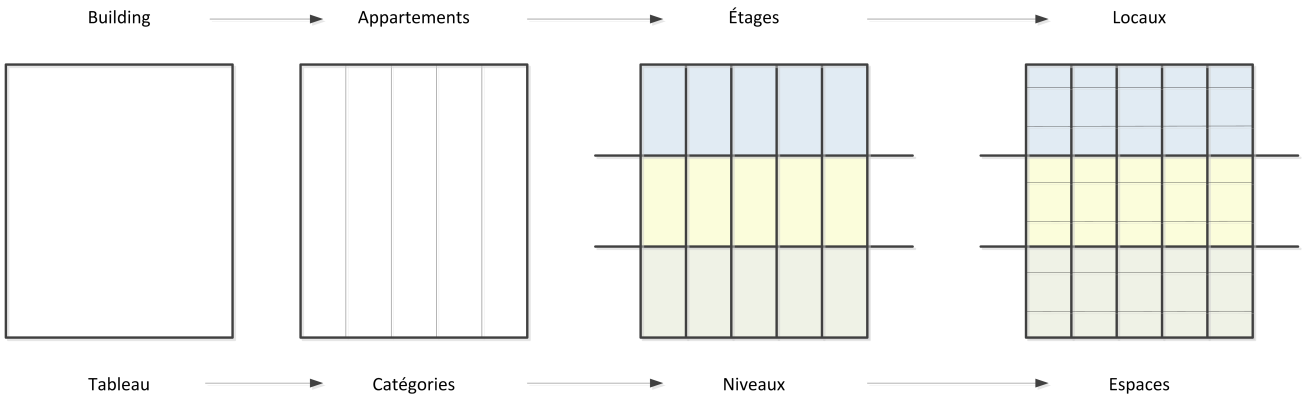

FIGURE 2.2 – Hiérarchie du tableau

2.1.2 Présentation des catégories

Les catégories sont déduites de faits observables : les ingénieurs sont des expérimentateurs par rapport aux réseaux. Ils sont à la fois praticien et théoricien. Ils doivent d'une part effectuer des mesures sur le réseau et d'autre part en déduire des lois générales. C'est sur base de ces mesures et déductions qu'ils peuvent travailler. Ces orientations théorique et pratique se retrouvent dans les générateurs de trafic. Par orientation théorique on entend des outils aidant à la conception intellectuelle méthodique et organisée des réseaux. Par orientation pratique on entend des outils permettant d'obtenir des résultats par l'expérimentation. Les générateurs de trafic sont répartis dans cinq catégories. Chaque catégorie contient un ensemble de générateurs prévus pour répondre à un même scénario donné. Nous décrivons pour chacune des catégories ce scénario. Nous donnons aussi les différences entre ces catégories. Précisons d'abord le vocabulaire : nous faisons la distinction entre simulation et émulation. Ces deux concepts se basent sur la notion de système. Un système est une collection organisée d'éléments soit théoriques soit physiques. Ces éléments collaborent ensemble partant de données en entrée pour produire un résultat en sortie. L'ensemble des éléments de la collection d'un système définit l'état interne de ce système. Un exemple type est une sucrerie. Il s'agit d'une usine composée de différents départements qui collaborent ensemble partant de betteraves en entrée pour produire du sucre en sortie. Ainsi, une émulation est la reproduction d'un système sur base de ses entrées et sorties uniquement. Pour des données identiques en entrée au système, une émulation produit les mêmes sorties que le système. Elle peut s'y substituer mais elle n'est pas forcément identique à l'état interne de ce système. Une simulation est une représentation artificielle formelle et explicite d'un système. Elle correspond le plus fidèlement possible à son état interne et permet l'analyse de celui-ci.

2.1.2.1 Analyse et simulation

Ces outils offrent un environnement de simulation. L'utilisateur va pouvoir recréer un réseau artificiel dans un environnement totalement maîtrisé. Ce réseau artificiel est caractérisé par deux concepts. Premièrement sa représentation. L'utilisateur doit définir un modèle mathématique pour ce réseau. Il sera exprimé de manière formelle et explicite. Des paramètres additionnels peuvent aussi être ajoutés. Deuxièmement, le réseau artificiel correspond à des éléments du monde réel. Ce peut être un réseau existant, un projet à l'état papier ou l'extension d'un réseau existant. De ce fait, la simulation est capable de soit fonctionner en isolation, soit fonctionner en interaction avec un réseau existant. Le modèle qui définit le réseau artificiel doit être suffisamment précis pour permettre ces interactions. Par interaction, on entend des échanges de trafic réseau. La simulation est constituée par : le modèle mathématique spécifié, l'ensemble des paramètres et l'environnement du logiciel de simulation. La figure 2.3 illustre le procédé de ces générateurs.

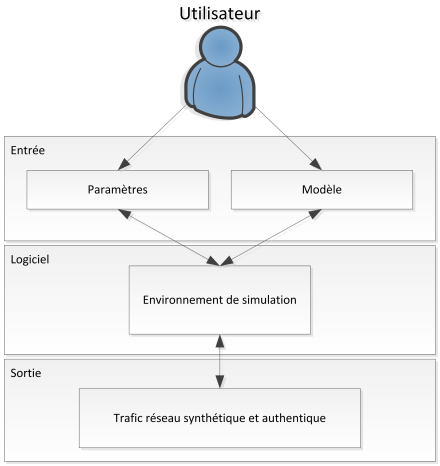

FIGURE 2.3 – Vue conceptuelle des outils d'analyse et de simulation

2.1.2.2 Génération modélisée

Ces logiciels effectuent une génération de trafic réseau sur base de modèles mathématiques. À la différence des outils de simulation, l'utilisateur ne peut pas créer son propre modèle. Il doit utiliser ceux proposés par le logiciel. Ces générateurs peuvent contenir autant de modèles qu'ils ne produisent de trafics différents. Ces différents trafics sont par nature synthétiques : ils proviennent tous d'une construction mathématique. Ce sont des émulateurs. Le trafic qu'ils produisent en sortie est conforme à celui produit par un système donné. En revanche, la façon dont est produit ce trafic ne l'est pas forcément. Ils sont typiquement utilisés pour vérifier si un réseau existant est apte à satisfaire au déploiement d'une nouvelle application. Chaque application produit une forme de trafic réseau qui lui est propre. Les générateurs basés sur des modèles permettent de recréer ce trafic. La figure 2.4 illustre le procédé de ces générateurs.

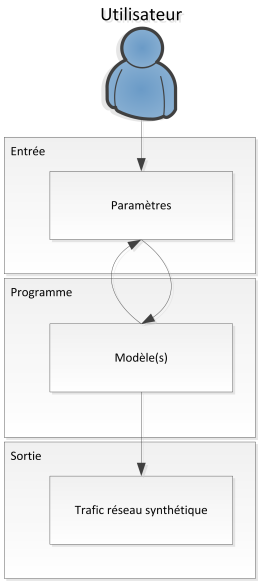

FIGURE 2.4 – Vue conceptuelle des outils de génération modélisée

2.1.2.3 Génération de montée en charge

Les générateurs de montée en charge sont des applications permettant d'augmenter l'intensité du trafic sur un réseau existant. On appelle cela effectuer une montée en charge. Ces applications permettent d'établir la robustesse d'un réseau face à une quantité d'informations plus importante à traiter. Pour y parvenir, elles décomposent le travail en cinq étapes :

1. L'utilisateur fournit ses entrées. Il s'agit en général de paramètres généraux mais certains outils acceptent aussi une trace réseau. Une trace réseau est un enregistrement de trafic réseau. Plus marginalement selon S. Molnar, certains outils offrent en plus la possibilité de se baser sur des mesures en temps réel pour définir automatiquement leurs paramètres d'entrée (voir [76]).
2. Les applications lancent une phase de prétraitement : ils effectuent une déduction d'informations statistiques sur les entrées. Cela leur donne les caractéristiques du trafic à produire.
3. En fonction de ces caractéristiques les applications vont dynamiquement adapter le traitement de l'étape suivante. Cette action est appelée auto-configuration dans la littérature.
4. Les applications vont utiliser des mécanismes de mise à l'échelle. Ce sont ces mécanismes qui sont dynamiquement sélectionnés à l'étape 3. Ils vont permettre de produire la montée en charge sur le réseau. Le trafic produit possède les mêmes caractéristiques statistiques que la trace réseau spécifiée en entrée et est conforme aux paramètres de l'utilisateur.
5. Le trafic ainsi produit est synthétique : il provient uniquement des modèles mathématiques des générateurs. Ces modèles mathématiques sont dits de "haut niveau". Ils couvrent les étapes deux à quatre. La figure 2.5 illustre le procédé de ces générateurs.

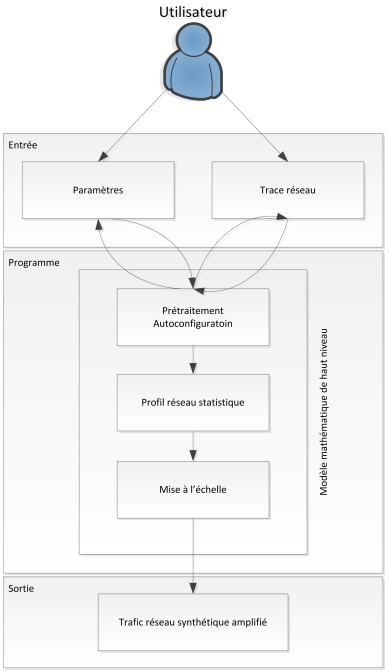

FIGURE 2.5 – Vue conceptuelle des générateurs de montée en charge

2.1.2.4 Génération de débit maximum

Ces logiciels ont pour objectif de produire du trafic réseau avec la plus forte intensité possible. Plus l'intensité est forte, plus la quantité de données en transit sur le réseau est importante. Ce type de générateur cherche à saturer un canal de communication pour déterminer le débit de données maximum que ce canal peut supporter. Il existe différents types de trafic réseau. Le débit maximum est influencé par le type de trafic réseau en transit. De ce fait, Il existe deux familles d'applications. La première regroupe les applications qui offrent une mesure "brute" du débit sans autres fonctionnalités à l'utilisateur. Ils ne sont pas configurables. La seconde famille est constituée des applications paramétrables. Elles laissent le choix à l'utilisateur de spécifier le type de trafic à générer. La figure 2.6 illustre le procédé de ces applications.

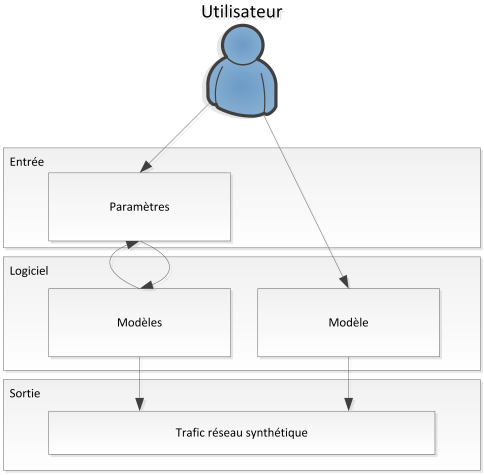

FIGURE 2.6 – Vue conceptuelle des générateurs de débit maximum

2.1.2.5 Moteurs de relecture

Un moteur de relecture fonctionne comme un radiocassette. Un radiocassette est capable de lire une cassette pour jouer la musique qu'elle contient. De la même manière, un moteur de relecture utilise un procédé équivalent. Il prend en entrée une trace réseau pour émettre sur le réseau le contenu de cette trace. Ces logiciels ne sont pas basés sur des modèles mathématiques pour la génération du trafic réseau : ils émettent des données contenues dans une trace. Plus marginalement, certains moteurs de relecture offrent des fonctionnalités complémentaires. Nous en avons trouvé deux dans notre revue de la littérature. Premièrement des fonctionnalités d'enregistrement pour capturer une trace réseau. Deuxièmement, la trace réseau à rejouer peut posséder un débit de données trop élevé pour un ordinateur. Ainsi, certains logiciels offrent la possibilité de diviser le travail de relecture entre plusieurs ordinateurs. La figure 2.7 illustre le procédé de ces applications.

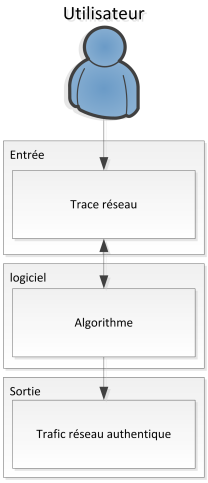

FIGURE 2.7 – Vue conceptuelle des moteurs de relecture

2.1.3 Structure d'une catégorie

Nous présentons l'architecture des catégories. Elles disposent toutes de la même structure. Soit trois niveaux de trois espaces. Nous parlerons des niveaux, ensuite des espaces. Les niveaux définissent les données produites en sortie par les générateurs. Les espaces définissent la vitesse à laquelle sont produites ces données en sortie. Pour un rappel de la structure du tableau, consulter la sous-section 2.1.1 situé page 6.

2.1.3.1 Les niveaux

Partons d'un cas de figure : Alice et Bob discutent. Il y a un dialogue entre eux. Ils échangent des phrases tour à tour. Une phrase est faite de mots. On peut dire qu'une phrase est un flux de mots. Un mot est une quantité unitaire d'information. Pris seul, il n'a pas forcément de sens. Mais un ensemble de mots groupés entre eux suivant un certain ordre en ont. Alice et Bob utilisent une langue pour parler. Elle peut être connue de tous : Français, Anglais, Espagnol, etc. Mais elle peut aussi être connue seulement d'eux même : langue secrète, jargon, etc. Il faut aussi considérer le niveau de langage. Ainsi, un avocat sait comment parler dans un tribunal. Il le fait suivant l'art et la manière nécessaire. C'est une application spécifique de la langue. Il se

situe à un niveau supérieur au niveau de la langue courante : il parle un jargon intellectualiste non compréhensible de tous. En revanche, un quidam ne sera pas capable de cela. Il parle une langue d'un niveau général. Chacun joue son rôle de manière cloisonnée.

D'un point de vue similaire, des logiciels peuvent communiquer entre eux sur un réseau. La langue qu'ils utilisent s'appelle un protocole. Un protocole connu de tous est dit libre. Un protocole uniquement connu de certains est dit propriétaire. Nous avons d'une part pour les protocoles libres : Transmission Control Protocol abrégé TCP, User Datagram Protocol abrégé UDP et Internet Protocol abrégé IP. Et d'autre part pour les protocoles propriétaire : MSN, YouTube et Skype. Ces énumérations sont non exhaustives. Lorsque deux logiciels se parlent, ils n'échangent pas à proprement parler des phrases ou des mots. Ils échangent des flux de paquets ou des paquets. Un flux de paquets peut contenir soit des paquets dont la charge de données utile est généraliste soit des paquets dont la charge de données utile est spécifique d'une application. On peut en déduire trois niveaux de dialogue : flux de paquets d'une application spécifique, flux de paquets généraliste et paquet. À ces trois niveaux de dialogue correspondent les trois niveaux associés à une catégorie. Ils permettent d'ordonner les générateurs suivant ce qu'ils sont capables de produire en sortie. Le tableau de la figure 2.8 présente l'intitulé de chaque niveau, son code couleur associé et son contenu. Nous présentons en annexe les différences de modélisation associées à chaque niveau (voir annexe C page 109). Cette découpe en niveau est présentée dans un article de Alessio Botta, Alberto Dainotti et Antonio Pescapé. Les références sont disponibles en [19].

Intitulé	Code couleur	Contenu
Niveau Application	Bleu	Générateurs produisant des flux de paquets d'une application spécifique
Niveau flux	Jaune	Générateurs produisant des flux de paquets généralistes
Niveau paquets	Vert	Générateurs produisant des paquets

FIGURE 2.8 – Tableau récapitulatif de la découpe en niveaux

2.1.3.2 Les espaces

Nous avons considéré les différents types de données produits par les générateurs de trafic IP. Cependant, il reste encore un paramètre à considérer. La vitesse à laquelle sont produites ces données. Cette vitesse s'exprime comme la quantité de données produite par unité de temps. Il s'agit d'un débit. Prenons un exemple. Soit trois étudiants : Alice, Bob et Charlie. Ils se livrent à un jeu dans un cercle de l'université. Un comptoir est recouvert de gobelets. Les étudiants sont placés à une distance variable de ce comptoir. Ils cherchent à récupérer le plus de gobelets possible. La durée d'un aller-retour de leur position au comptoir est de 30 secondes pour Alice, 15 secondes pour Bob et 4 secondes pour Charlie. Les étudiants peuvent aller chercher des gobelets à raison de deux par aller-retour. Le schéma de la figure 2.9 illustre cette situation. La position des étudiants est définie par des cercles concentriques. Ces cercles définissent trois

espaces. Pour chaque espace, la durée d'un aller-retour est constante. Il y a un étudiant par espace. Comme les durées d'aller-retour varient en fonction des espaces, on peut dire que les espaces définissent des priorités d'accès entre les étudiants.

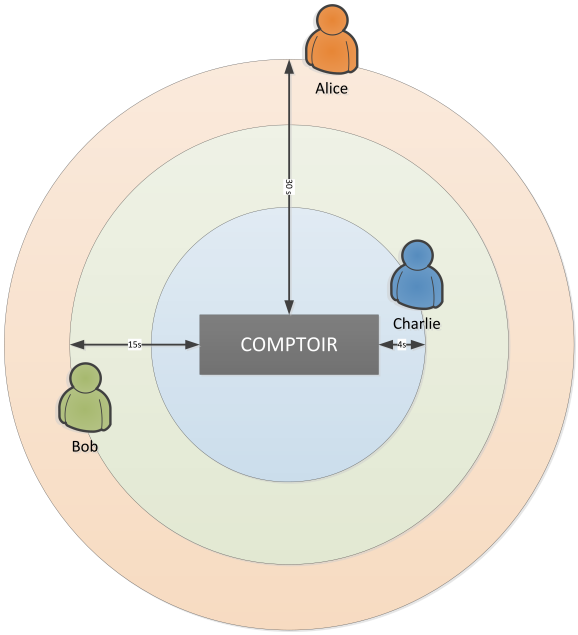

FIGURE 2.9 – Un jeu universitaire atypique

Connaissant la durée d'un aller-retour, on définit la fréquence d'accès au comptoir pour chaque espace. Nous exprimons cette fréquence comme le nombre maximum d'accès au comptoir par minute :

— Fréquence d'accès de l'espace d'Alice : $60 \div 30 = 2$
— Fréquence d'accès de l'espace de Bob : $60 \div 15 = 4$
— Fréquence d'accès de l'espace de Charlie : $60 \div 4 = 15$

Elle signifie que dans le meilleur des cas, Alice fera deux accès au comptoir, Bob quatre accès et Charlie quinze accès. Sur base de ces fréquences, on peut calculer un débit qui exprime le nombre de gobelets récupérés par minute. On note ce débit G/m :

— Alice : $2 \times 2 = 4\ G/m$
— Bob : $2 \times 4 = 8\ G/m$
— Charlie : $2 \times 15 = 30\ G/m$

Les espaces définissent des fréquences. Les fréquences définissent des débits. Alors par une relation de transitivité entre les espaces, les fréquences et les débits on dira que les espaces définissent des débits. Considérons maintenant que le cercle universitaire dans lequel jouent les étudiants est un système d'exploitation moderne et que les étudiants sont des logiciels. Posons le vocabulaire : Premièrement, un fichier est un ensemble de données regroupées sous le même nom, à l'image d'une boite. Pour la définition d'un logiciel et d'un programme, consulter la section 2.1 page 3. Ainsi, pour qu'un logiciel produise des données, il doit exécuter ses programmes.

Le composant physique de l'ordinateur qui exécute les instructions d'un programme s'appelle le processeur. Deuxièmement, un système d'exploitation est une couche logicielle faisant le lien entre le matériel physique de l'ordinateur et les applications qui y sont installées. Typiquement : Microsoft Windows®, Mac OS®, GNU/Linux, etc. Ces systèmes sont dits multitâches : ils sont capables de donner l'illusion que plusieurs logiciels s'exécutent au même instant en partageant l'accès au processeur entre les programmes des logiciels. Ces programmes s'exécutent tour à tour durant un laps de temps suffisamment court. Cela donne l'illusion d'une exécution en parallèle de l'ensemble des logiciels à l'utilisateur. Pour y parvenir, les systèmes d'exploitation modernes définissent des priorités entre les programmes à exécuter. À ces priorités correspondent des espaces dans lesquels les programmes sont placés. La situation est analogue à celle des étudiants dans le cercle universitaire. On distingue trois espaces : l'espace utilisateur abrégé E_U, l'espace noyau abrégé E_N et l'espace physique abrégé E_P. Comme pour les étudiants, ces espaces définissent des temps d'accès au processeur exprimés sous forme de fréquence. Plus la fréquence d'accès au processeur est élevée, plus le débit de données qu'un logiciel peut produire est élevé. Si on exprime $F(E)$ comme la fréquence d'accès F de l'espace E au processeur, on obtient :

$$F(E_U) \leq F(E_N) \leq F(E_P)$$

De cette manière, un générateur de E_U aura un débit maximum plus faible qu'un générateur de E_N et un générateur de E_N aura un débit plus faible qu'un générateur de E_P. Le même raisonnement s'applique par transitivité aux générateurs de E_U et de E_P. Cette constatation est vraie si les générateurs s'exécutent sur un même ordinateur équipé du même système d'exploitation. Le choix de concevoir une application dans l'un ou l'autre de ces espaces dépend des objectifs à atteindre. Le développeur de l'outil choisira le plus approprié pour son projet. Les trois paragraphes suivant expliquent les raisons pouvant motiver le choix de l'un ou l'autre de ces espaces.

2.1.3.3 Espace utilisateur

Les générateurs de l'espace utilisateur se présentent sous la forme d'applications conventionnelles. Ils sont disponibles sous la forme de code source et de fichiers binaires. Ce sont des applications multi-plateformes. Elles sont majoritaires dans notre échantillon. Elles représentent 21 applications sur les 27 étudiées. Soit 78 %. La complexité de développement d'un tel générateur est identique à celle d'une application ordinaire. Le développeur bénéficie de la souplesse offerte par les architectures modernes. Il peut recourir à des opérations et services préconçus pour simplifier le traitement et la génération du trafic. En effet, le développeur peut utiliser des interfaces de programmation offertes par le systèmes d'exploitation et des librairies dédiées. Une interface de programmation désigne un ensemble normalisé d'opérations et de services offerts par un logiciel à d'autres logiciels.

D'une part, le système d'exploitation fournit une interface de programmation régie par le standard « Portable Operating System Interface » abrégé POSIX. C'est un ensemble de normes définies par la IEEE Computer Society. Elles ont pour objectif de maintenir la compatibilité entre les systèmes d'exploitation (voir [142]). Un développeur s'appuyant sur la spécification POSIX a l'assurance de pouvoir porter son application sur un autre système d'exploitation sans effectuer de modifications majeures. En effet, le générateur interagit avec le système d'exploitation uniquement au travers de son interface de programmation : il dépend seulement du

standard POSIX. D'autre part, il existe des librairies de programmation spécialisées. Parmi ces bibliothèques, on trouve notamment "LibCap / TCPDump" et "HPCAP". Elles sont utilisées pour la capture de trafic (voir respectivement [126] et [138]). Les bibliothèques "Libcrafter" et "Scapy" sont spécialisées dans la création et l'analyse de paquets sur le réseau (voir respectivement [127] et [135]). Scapy et Libcrafter sont disponibles sous la forme d'une boite à outil intégrant plusieurs utilitaires. Ces utilitaires permettent de manipuler plusieurs protocoles et d'analyser des flux de paquets capturés ou enregistrés dans une trace. L'ensemble de ces quatre bibliothèques est rédigé en langage C++. Le développement orienté objet permet de gérer plus aisément la complexité de l'application par rapport aux deux autres espaces suivants. La richesse d'opérations et de services fournis par les architectures modernes permet la conception de générateurs de trafic portables et dotés d'un grand nombre de fonctionnalités.

La fréquence d'accès au processeur qui caractérise l'espace utilisateur est limitée par la nécessité de devoir faire appel aux opérations et services du systèmes d'exploitation et des bibliothèques. Le générateur ne peut pas générer de trafic sur le réseau sans utiliser ces fonctionnalités. En effet, une application ne peut pas s'autoriser l'accès exclusifs aux ressources de l'ordinateur. Elle doit passer par l'interface de programmation du système d'exploitation pour y accéder.

2.1.3.4 Espace noyau

Les générateurs de l'espace noyau sont des applications divisées en deux logiciels. Premièrement, un logiciel situé dans l'espace utilisateur. Ce logiciel permet à l'utilisateur d'introduire ses paramètres et de lancer l'ordre de génération de trafic. Deuxièmement, un logiciel situé dans l'espace noyau. Ce logiciel est un module. Il se greffe au sein du système d'exploitation. Ils est exécuté comme les propres composants du système d'exploitation. C'est vers ce module que les paramètres de l'utilisateur sont transférés et traités. Un module situé dans l'espace noyau permet d'avoir une plus grande priorité d'accès aux ressources de l'ordinateur. Ce mécanisme augmente le débit maximum des applications situées dans cet espace.

Les générateurs s'exécutant dans l'espace noyau ne sont pas portables. Ils sont intrinsèquement liés aux systèmes d'exploitation pour lesquels ils ont été conçus. Leur processus d'installation est également plus complexe. Des connaissances en programmation système sont nécessaires pour concevoir ce type de générateur. La programmation d'un module exige la maîtrise du langage C et des connaissances techniques sur le fonctionnement interne d'un système d'exploitation. L'espace noyau est situé plus bas que l'espace utilisateur. Cela a des conséquences sur les opérations et services offerts au développeur. En effet, il ne peut pas utiliser de bibliothèques spécialisées pour le module. Il doit se baser exclusivement sur l'interface de programmation offerte par le système d'exploitation. Le logiciel NTGen est un exemple (voir [130]). L'interaction entre les deux logiciels peut nécessiter l'interfaçage entre les langages objets et impératifs tels que les langages C et C++. Notre échantillon contient quatre générateurs de ce type. Il s'agit de NTGen, Kute, TCPivo et PKTGen. Les références des articles liés à ces générateurs sont respectivement accessibles en [130], [155], [39] et [82]. Pour tous, l'ensemble des fonctionnalités est inférieur à celui des générateur situés dans l'espace utilisateur.

La fréquence d'accès au processeur qui caractérise l'espace noyau est limité par la nécessité de devoir faire appel aux opérations et services du système d'exploitation. Les bibliothèques

spécialisées ne sont pas utilisables dans le module. Le trafic généré aura un débit maximal plus important grâce à une priorité d'accès plus élevée aux ressources de l'ordinateur .

2.1.3.5 Espace physique

Les générateurs de l'espace physique sont des applications assistées matériellement. Pour rappel, la définition d'une application est donnée à la section 2.1 page 3. Ces générateurs sont spécifiquement conçus pour s'exécuter avec un ou plusieurs périphériques dédiés. Ils effectuent une séparation entre le quoi et le comment du trafic à générer. Le quoi désigne le trafic à produire. Il est géré par l'application. Ce trafic est déterminé automatiquement ou selon les paramètres de l'utilisateur. Le comment désigne les opérations à réaliser pour obtenir le trafic spécifié. Cette tâche est confiée à un périphérique dédié. Il n'est plus nécessaire d'utiliser l'interface de programmation du système d'exploitation. La génération de trafic est réalisée exclusivement par le périphérique dédié. Les générateurs IXPKTGEN et Bruno s'exécutent dans l'espace physique. Les références des articles liés sont disponibles en [16] et [8]. Dans ces articles, les auteurs décrivent l'utilisation d'une carte réseau de la gamme Intel IXP 2400. Ces périphériques sont directement programmés en langage C. La documentation technique décrit l'ensemble des fonctionnalités (voir [30]).

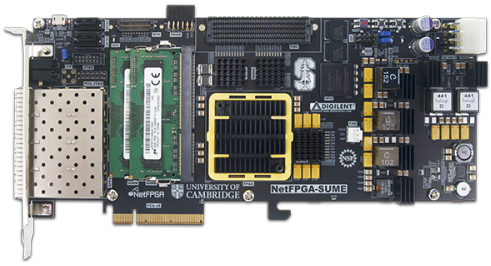

FIGURE 2.10 – Plateforme Réseau NetFPGA version *"SUME"* en connexion PCI Express

Dans [31] Adam Covington décrit un générateur de trafic intitulé « The NetFPGA Packet Generator » Il s'agit d'un générateur basé sur la plateforme NetFPGA. Dans cet article, l'auteur expose l'architecture, l'implémentation et les expériences réalisées sur cet outil. Les fonctionnalités décrites permettent l'émission et l'enregistrement de traces réseau. L'expérience illustre une différence de précision entre les générateurs des espaces utilisateur et noyau par rapport aux générateurs de l'espace physique. En effet, pour les espaces utilisateur et noyau, le délai entre l'émission de deux paquets n'est pas toujours constant. Il peut y avoir des variations sur ce délai. Cela affecte la précision que l'expérimentateur peut attendre de ces générateurs. Pour les générateurs de l'espace physiques reposant sur le projet NetFPGA, l'auteur illustre deux faits par l'expérience. Premièrement, le délai entre les paquets est constant sur l'ensemble du trafic généré. Deuxièmement, le débit d'émission en sortie est conforme aux paramètres de l'utilisateur. NetFPGA est un projet de l'université de Cambridge développé en partenariat avec des entreprises privées. Nous avons découvert cette plateforme lors d'une conférence donnée par Victor Ucet (voir [122] et [95]). Dans [47]et [67] John Lockwood définit NetFPGA comme : *"une plateforme permettant aux étudiants et aux chercheurs de construire des systèmes réseaux à haute performance"*. Il s'agit d'une carte réseau aux fonctionnalités avancées. Une description complète de ses fonctionnalités et de ses cas d'utilisation potentiels sont disponibles dans [157].

La figure 2.10 présente une illustration de cette carte réseau [1]. La société "Digilent Inc" met à disposition cette plateforme sur son site internet [2]. En date du 19 juillet 2015, elle est accessible au prix public de 9 750 $ et à un tarif préférentiel de 4 995 $ pour le domaine académique.

Les générateurs de l'espace physiques permettent d'atteindre des débits maximums plus élevés que les deux catégories précédentes. Ainsi, la spécification de la plateforme NetFPGA stipule un débit maximum de 100 Gb/s (voir [157]). La gamme Intel IXP 2400 offre des performances plus modestes. La fiche technique indique un débit maximal de 2,5 Gb/s (voir [26]). Ils offrent également une plus grande régularité dans le trafic généré (voir [67]). Ces générateurs sont complexes à concevoir. Ils exigent des connaissances techniques pointues sur le matériel utilisé. Le générateur sera systématiquement dépendant de ce matériel. En conséquence, les générateur de l'espace physique ne sont pas portables. En outre, se pose la question de l'accessibilité : Il faut pouvoir se procurer le matériel. Le prix est un premier obstacle. La disponibilité du matériel en est une second.

2.2 Choix d'un générateur de trafic

Nous avons vu à la section 2.1 page 3 qu'il existait différentes catégories de générateurs. À ces catégories correspondent des utilisations types. Ensuite, nous avons identifié trois niveaux définissant le type de trafic produit en sortie par ces logiciels. Enfin, nous avons présentés trois espaces : ils définissent les performances avec lesquelles le trafic est produit. Comme nous l'avons annoncé dans l'introduction de ce chapitre page 3, notre objectif est de sélectionner un générateur de trafic pour notre expérience du chapitre 5 page 43. Nous avons sélectionné le générateur de trafic D-ITG qui signifie *"Distributed Internet Trafic Generator"*. Nous l'avons sélectionné pour cinq raisons principales :

Représentativité :
D-ITG est un générateur de l'espace utilisateur placé au sein des catégories "génération modélisée" et "génération de montée en charge". Ces deux catégories contiennent 17 générateurs sur les 27 de notre échantillon, soit 62 %. Comme nous l'avons évoqué dans la sous-section 2.1.3.3 page 13, les générateurs de l'espace utilisateur représentent 78% des générateurs de notre échantillon. En ce sens, D-ITG est un outil représentatif de l'ensemble des générateurs considérés.

Accessibilité :
C'est un générateur gratuit, open-source et multi-plateforme. Il est disponible sous licence GPL. GPL est l'acronyme de *"General Public Licence"*. C'est une licence juridique décrivant les conditions légales de distribution et d'utilisation d'un logiciel libre. L'application est disponible sous forme de code source. Elle doit être compilée et installée pour être utilisée. Ce statut rend le générateur accessible à tous sans restriction.

Popularité :
D-ITG est un outil bien référencé dans la littérature. Il est donc connu de la communauté scientifique. Une recherche avec l'outil "Google Scholar" nous donne 182 citations en date du 19 juillet 2015 pour l'article de référence de ce générateur (voir [20]).

1. http://digilentinc.com/Data/Products/NETFPGA-10G-SUME/NetFPGA-SUME-top-600.png
2. http://digilentinc.com

Documentation :
Le projet responsable de D-ITG est toujours actif en date de juillet 2015. Le site internet propose un manuel d'utilisation détaillant l'ensemble des fonctionnalités et leurs utilisations. Il peut être consulté en ligne ou téléchargé au format PDF (voir [10]).

Performances :
La comparaison entre D-ITG et d'autres générateurs de trafic montre une plus grande performance de ce générateur en particulier. Nous exposons précisément cette notion au point F page 115.

2.3 Cahier des charges

Cette section présente le cahier des charges du générateur D-ITG. Les notions de documentation et de performance proviennent du cahier des charges. Dans le cadre de D-ITG, cinq documents décrivent l'application. Il existe un seul cahier des charges mais quatre documents basés sur son contenu. Il convient de ne pas les confondre. Pour chacun d'eux, nous donnons une brève description du contenu :

Cahier des charges complet :
L'introduction expose les idées, concepts et réflexions ayant conduit à la réalisation de D-ITG. Ensuite, le document explique ce qu'est D-ITG et les aspects principaux de sa conception. Après, l'architecture logicielle est exposée en détails. D'une part, les mécanismes mis en oeuvre pour la génération de trafic. D'autre part, l'ensemble des logiciels composant l'application D-ITG. Tous ces éléments sont commentés et illustrés à l'aide de diagrammes. Les auteurs utilisent trois types de diagrammes :
— Diagramme machine à états
— Diagrammes de séquence UML
— Diagrammes d'activité UML
S'ensuivant, les auteurs illustrent les performances de D-ITG. Le cahier des charges contient une évaluation des performances et une analyse comparative avec d'autres générateurs de trafic. La référence de ce document est disponible en [12].

Description de l'architecture :
Ce document apparaît comme un résumé du cahier des charges. D'abord une description générale de D-ITG et de ses fonctionnalités principales. Ensuite, une description allégée de l'architecture. Les diagrammes d'activité et de séquence n'apparaissent plus. La description des logiciels est plus succincte. Après, l'article contient une présentation de deux expériences possibles. La première illustre le contexte des réseaux sans fils. La seconde illustre une déploiement sur le réseau internet. En annexe, on retrouve un tableau comparatif des fonctionnalités de dix générateurs. La référence de l'article est disponible en [11].

Article de présentation de D-ITG :
L'article expose d'abord les générateurs de trafic existant avant D-ITG. Les auteurs décrivent leurs fonctionnalités et justifient la nécessité d'un nouveau générateur : ils proposent D-ITG. Ensuite, ils décrivent l'architecture et les fonctionnalités de l'application. Après quoi, ils justifient l'apport de cet outil à la communauté scientifique. Enfin, un en-

semble de résultats expérimentaux dérivés du cahier des charges complet sont présentés. La référence de l'article est disponible en [20].

Manuel d'utilisation :
D'abord une explication générale de l'architecture. Ensuite, la liste de fonctionnalités et les détails nécessaires à leur manipulation. Enfin, quelques exemple de difficulté progressive pour l'utilisation de l'application. La référence du manuel est disponible en [10].

Support de présentation de D-ITG :
Ce support de présentation fut utilisée par l'équipe de recherche pour présenter D-ITG lors des conférences. Premièrement, des explications sur les motivations du projet, l'origine des réflexions et des idées de conception. Deuxièmement, une liste des cas d'utilisation potentiels du logiciel. Ces cas d'utilisation sont décrits dans l'introduction du cahier des charges. Ensuite, exposition de l'architecture et illustration d'une utilisation potentielle. Après quoi, présentation d'une analyse comparative avec d'autres générateurs de trafic. Les graphiques de cette analyse sont tirées du cahier des charges complet. La référence du support de présentation est disponible en [21].

Le cahier des charges contient deux catégories d'informations. Premièrement, une description de l'architecture et des fonctionnalités du générateur de trafic. Deuxièmement, des résultats expérimentaux illustrant ses performances. Nous présenterons dans les sections suivantes les aspects importants du cahier des charges. L'ensemble des détails peut être consulté dans [12]. Nous exposerons dans la sous section 2.3.1 la première partie du cahier des charges. Nous expliquerons la seconde partie du cahier des charges dans la sous-section F page 115. Nous y présenterons les résultats expérimentaux menés par l'équipe de recherche et développement pour illustrer les performances de D-ITG par rapport à d'autres générateurs de trafic. Nous évoquerons également les performances de D-ITG sur différents systèmes d'exploitation.

2.3.1 Présentation générale

Cette sous section donne un aperçu général du générateur D-ITG. Nous détaillerons d'abord ses cas d'utilisation potentiels. Ensuite, nous exposerons ses fonctionnalités. Enfin, nous expliquerons son architecture.

2.3.1.1 Cas d'utilisations potentiels

D-ITG est pertinent pour plusieurs cas d'utilisation potentiels. Ces informations proviennent du support de présentation (voir [21]).

- **Analyse de réseaux hétérogènes :**
  - Réseaux câblés
  - Réseaux sans fils
  - Réseaux mobiles

- **évaluation des performances selon quatre métriques :**
  - Le débit
  - La perte de paquets
  - Le délai de transmission :

  - "Round trip time" abrégé RTT : durée de temps nécessaire à un paquet pour réaliser un aller-retour entre un émetteur et un récepteur au travers d'un réseau.
  - "One way delay" abrégé OWD : durée de temps nécessaire à un paquet pour réaliser un aller simple entre un émetteur et un récepteur au travers d'un réseau.
- La gigue : La gigue est la variation de la latence au cours du temps. Plus elle est élevée, plus les délais sont variables. Plus précisément selon [33] : " La gigue est la différence de délai de transmission de bout en bout entre des paquets faisant partie d'une même flux, sans prendre en compte les paquets perdus ".

- **Tester les capacités de différents périphériques :**
  - Stations de travail et serveurs informatiques
  - Ordinateurs portables, tablettes et assimilés
  - Smartphone et Pocket PC

- **Tester la qualité de service des architectures :** La qualité de service abrégée QoS désigne la performance du réseau tel qu'elle est perçue par les utilisateurs. C'est une mesure qualitative. Elle intègre des métriques tels que ceux évoqués précédemment. L'objectif de la QoS est d'exprimer l'expérience vécue par les utilisateurs avec un réseau. La mesure de la Qos intègre tous les aspects susceptibles d'avoir un impact sur cette expérience : latence, taux d'erreurs, durée moyenne entre deux pannes, etc.

- **Analyse des algorithmes de routage :** Le routage désigne l'ensemble des règles et opérations mises en oeuvre pour assurer la navigation correcte des paquets au sein d'un réseau. Un périphérique qui effectue du routage est nommé routeur.

- **Ingénierie du trafic réseau**

- **Analyse du comportement réseau**

- **Montée en charge**

•

2.3.1.2 Liste des fonctionnalités

La liste des fonctionnalités est extraite du manuel de D-ITG (voir [10]).

- **Propriétés personnalisables des flux générés :**
  - La durée du trafic généré en millisecondes
  - Le délai avant démarrage effectif de la génération de trafic en millisecondes
  - Le nombre total de paquets à envoyer
  - Le nombre total de Kilo octets à envoyer

- **Protocoles supportés :**
  - IPv4 et IPv6 :

  - Spécification des adresses IP source et de destination : Une adresse IP est une suite de quatre ou seize octets permettant d'identifier un hôte au sein d'un réseau.
  - Spécification de la valeur TTL : TTL signifie "Time To Live". C'est un nombre entier déterminant le nombre maximal de sauts qu'un paquet peut réaliser au sein d'un réseau. Le TTL est décrémenté à chaque saut jusqu'à atteindre zéro : à ce moment, le paquet est détruit. Ce mécanisme évite la circulation des paquets pour une durée infinie sur le réseau.
  - Transpercement de routeur NAT : un routeur NAT est un périphérique réseau qui effectue du "Network Adress Translating", soit de la traduction d'adresses IP vers d'autres adresses IP.
- TCP, UDP, ICMP, DCCP et SCTP : Spécification des numéros de port source et de destination. Un numéro de port est un nombre entier qui permet d'identifier une "porte d'entrée" sur un hôte réseau. Il permet de différencier les différents services disponibles sur un même hôte.

- **Profils d'applications réseaux prédéfinis :**
  - Telnet
  - DNS
  - Quake 3
  - Counterstrike (mode actif et passif)
  - VoIP (G.711, G.729 et G.723)

- **Définition de la charge des paquets :**
  - Charge définie aléatoirement
  - Charge définie depuis une lecture sur fichier

- **Définition des valeurs de PS et de l'IDT :** PS signifie "Packet Size". Il s'agit de la quantité de donnée transportée par un paquet. IDT signifie "Inter Departure time". C'est la quantité de temps entre l'envoie de deux paquets. La figure 2.11 illustre ces deux concepts [3].

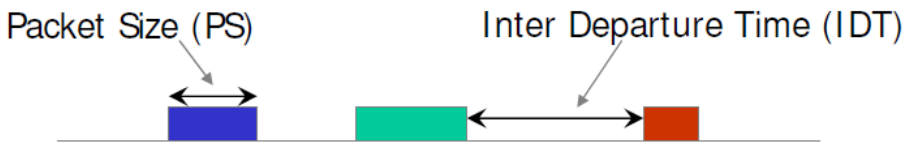

FIGURE 2.11 – Illustration de PS et de l'IDT

Trois ensembles de fonctionnalités permettent de définir les valeurs de PS et d'IDT :
- Distribution prédéfinies basées sur des variables aléatoires :
  - Uniforme
  - Constante

3. Graphique issu de [21]

  - Exponentielle
  - Cauchy
  - Pareto
  - Normal
  - Poisson
  - Gamma
  - Weibull
- Spécification explicite de la semence aléatoire utilisée pour la reproductibilité des flux générés
- Chargement des valeurs de PS et de l'IDT depuis une lecture sur fichier

- **Métriques pour la QoS :**
  - Débit en octets par seconde
  - Débit en paquets par seconde
  - Gigue en millisecondes
  - Perte de paquets
  - Délai :
    - Round trip time en millisecondes
    - One way delay en millisecondes

- **Stockage d'informations sur le trafic généré :** D-ITG stocke l'information du trafic généré sur l'émetteur et le récepteur. Cela permet une isolation des données relatives aux machines hôtes et au réseau.

- **Multi-flux :** L'application offre la possibilité d'émettre et de recevoir plusieurs flux réseaux simultanément.

2.3.1.3 Architecture

D-ITG est comparable à une boite à outils. C'est une application composée de cinq logiciels et d'un protocole propre à elle même. Chacun de ces logiciel possède une fonction spécialisée. Ils sont tous utilisables en ligne de commande. La figure 2.12 [4] illustre l'architecture de l'application. Nous allons maintenant décrire chacun des logiciels.

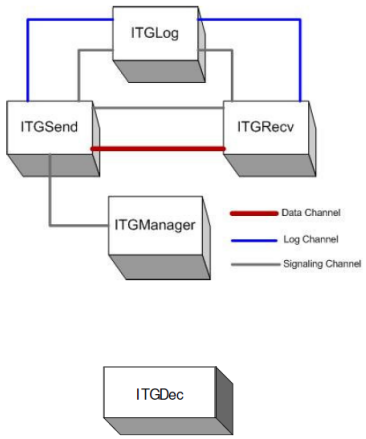

FIGURE 2.12 – Architecture de D-ITG

ITGSend :
ITGSend est le logiciel d'émission de trafic. Il peut être utilisé de deux manières. Premièrement, il peut être utilisé localement sur la machine hôte. Deuxièmement, il peut être piloté à distance par une instance de ITGManager. Le logiciel génère des informations sur les flux envoyé. Les données collectée se font à l'échelle des paquets. Ces informations peuvent être prise en charge par ITGLog. On appelle émetteur l'hôte réseau qui exécute une instance d'ITGSend.

ITGRecv :
ITGRecv est le logiciel de réception du trafic. Il attend en permanence une nouvelle connexion TSP. Lorsqu'une demande de connexion TSP survient, le logiciel négocie les paramètres de l'expérimentation avec l'hôte à l'origine de cette demande. Comme ITGSend, ITGRecv peut générer des informations sur les flux générés. Ici encore, les données collectées se font à l'échelle des paquets. Ces données peuvent être prises en charge par ITGLog. On appelle récepteur un hôte réseau qui exécute une instance d'ITGRecv.

ITGLog :
ITGLog est le logiciel assurant le stockage des données collectées par ITGSend et ITGRecv. Ces données regroupent les sept informations suivantes pour chaque flux généré :
— Identifiant du flux : un nombre naturel
— Numéro de séquence : un nombre naturel
— Adresse IP de l'émetteur
— Adresse IP du récepteur
— Durée de transmission en secondes
— Durée de réception en secondes
— Taille des paquets en octets

4. Graphique issu de [21]

Ces données sont stockées sous forme de fichier Log brut. Elles ne sont pas directement exploitables. L'objectif d'ITGLog est de stocker ces informations le plus vite possible pour en limiter l'impact sur le trafic généré. Ces fichiers Log peuvent être stockés localement sur la machine hôte ou à distance sur le réseau. Les données seront analysées postérieurement par ITGDec.

ITGDec :

ITGDec est le logiciel générant les résultats d'analyse à partir des informations collectées par ITGLog. Il permet d'obtenir un ensemble de données statistiques :

- Débit moyen :
  - Débit exprimé en kilo octet par seconde
  - Débit exprimé en paquets par seconde
- Délai minimum, maximum et moyen :
  - One way Delai en milliseconde
  - Round Trip Time en milliseconde
- Gigue en milliseconde
- Perte de paquets

ITGDec permet d'obtenir les valeurs pas seulement pour la durée totale d'un flux mais aussi pour une portion de ce flux.

ITGManager :

ITGManager est un logiciel de contrôle à distance pour ITGSend, ITGRecv et ITGLog. Il permet de contrôler de multiples instances de ces logiciels disséminées au sein d'un réseau. Le contrôle est effectué au travers d'une interface de programmation. La figure 2.13 illustre un déploiement expérimental possible à l'échelle d'un réseau [5].

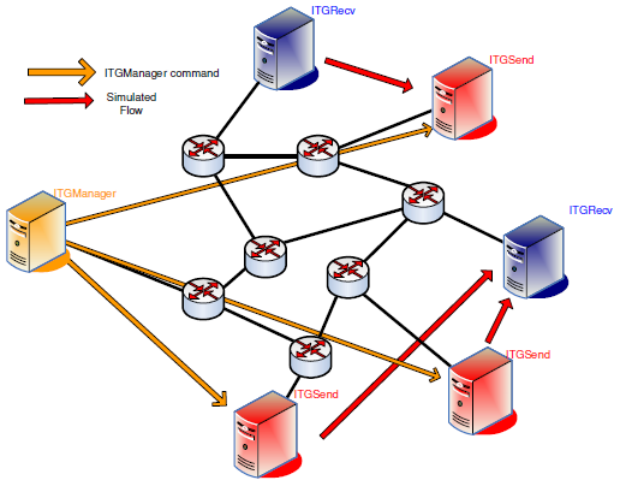

FIGURE 2.13 – Exemple d'expérience à l'échelle d'un réseau

TSP ou "Trafic Specification Protocol" :

TSP est un protocole spécifique à D-ITG. Il repose sur le protocole TCP. TSP permet la communication entre les différents logiciels. Il permet la définition des paramètres pour chaque expérimentation. Les instances de ITGSend et ITGRecv peuvent négocier les paramètres de l'expérimentation et le contrôle de la génération de trafic en utilisant ce protocole. TSP assure cinq responsabilités :

5. Graphique issu de [21]

— Création d'une connexion entre une instance de ITGSend et ITGRecv
— Fermeture d'une connexion entre une instance de ITGSend et ITGRecv
— Authentifier une instance de ITGRecv pour une instance de ITGSend
— Échange d'informations sur une génération de trafic
— Détection des événements issus d'une génération de trafic

Lorsque plusieurs flux transitent simultanément entre une instance de ITGSend et ITGRecv, une seule connexion TSP est utilisée entre les deux logiciels pour contrôler l'ensemble des flux.

2.3.2 Résultats expérimentaux

Les résultats expérimentaux illustrent les performances de D-ITG. Nous définirons d'abord la notion de performance telle qu'elle fut considéré par les auteurs du générateur. Ensuite, nous expliquerons le montage effectué pour la réalisation des tests. Enfin, nous présenterons l'analyse comparative et l'étude de performance.

2.3.2.1 Notion de performance

Les auteurs de D-ITG comparent les performances de leur générateur avec d'autres générateur existants. Dans ce contexte, la notion de performance exigent des comparaisons entre ces applications. Une comparaison objective ne peut se faire que sur des critères identiques. Dans le cas des générateurs, il convient de comparer des fonctionnalités identiques entre elles. Dans [21], Alessio Botta expose une comparaison des fonctionnalités de D-ITG et neuf autres générateurs connus. La figure 2.14 illustre le tableau de synthèse disponible dans cet article [6].

Traffic Generators	Operating Systems			Protocols						Options						Operative mode			Logging Phase			Meters Type	
	LINUX	Windows	Unix like	UDP	TCP	ICMP	Telnet	VoIP	DNS	TTL	TOS	Priority	Seed	Duration	Delay	Single Flow	Remote	Multiple Flow	Sender	Receiver	Remote	OWDM	RTTM
D-ITG	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
RUDE/CRUDE	✓		✓	✓								✓		✓		✓				✓		✓	
MGEN	✓	✓	✓	✓						✓	✓			✓		✓		✓		✓		✓	
TG2	✓		✓	✓	✓					✓	✓			✓		✓			✓	✓		✓	
Iperf	✓	✓	✓	✓	✓									✓		✓		✓				✓	
NetProbe	✓		✓	✓												✓		✓	✓				✓
TfGen		✓		✓												✓			✓				
Traffic	✓	✓	✓	✓	✓									✓		✓		✓					
MTOOLS	✓		✓	✓							✓		✓	✓	✓	✓		✓	✓	✓		✓	✓
UDPGenerator	✓		✓	✓												✓				✓		✓	

Traffic Generators	IDT								PS								
	Pareto	Constant	Uniform	Exponential	Cauchy	Normal	Gamma	Poisson	Pareto	Constant	Uniform	Exponential	Cauchy	Normal	Gamma	Poisson	Incremental
D-ITG	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
RUDE/CRUDE		✓								✓							
MGEN		✓						✓		✓							
TG2		✓	✓	✓						✓	✓	✓					
Iperf		✓								✓							
NetProbe		✓								✓							
TfGen		✓								✓							
Traffic		✓	✓							✓	✓						✓
MTOOLS	✓	✓	✓	✓					✓	✓	✓	✓					
UDPGenerator		✓								✓							

FIGURE 2.14 – Comparaison des fonctionnalités entre dix générateurs de trafic

6. Tableau issu de [21]

La seule fonctionnalité commune entre ces générateurs est l'émission d'un trafic UDP à débit constant. En conséquence, c'est le type de trafic utilisé pour l'analyse comparative et l'analyse des performances. C'est le seul à permettre la comparaison des générateurs les uns par rapport aux autres. Les auteurs de D-ITG font le choix de comparer les débits maximum de plusieurs générateurs selon ce type de trafic.

2.3.2.2 Montage expérimental

Nous donnons l'ensemble des informations disponibles dans le cahier des charges (voir [12]). Nous n'avons trouvé aucune information supplémentaire décrivant assez précisément le montage effectué pour réitérer l'expérience : certaines informations essentielles sont manquantes. Le montage se compose de deux ordinateurs connectés l'un à l'autre par un câble réseau. La norme de ce câble n'est pas indiquée. Ce câble possède une connexion RJ45 à chacune de ses extrémités. La figure 2.15 illustre les spécifications expérimentales disponibles dans le cahier des charges [7].

Hardware Details	Intel Pentium 4 2,6 GHz - CPU Cache 512
	RAM: 1024 MB
	Hard Disk: Maxtor 6Y080L0 (Fast ATA/Enhanced IDE Compatible, Ultra ATA/133 Data Transfer Speed, 2MB Cache Buffer, Quiet Drive Technology, 100% FDB motors)
Networks Details	2 PCs with a Gigabit Ethernet back-to-back connection
	Ethernet Controller: 3Com Gigabit LOM (3c940)
Software Details	**Linux:** Linux Mandrake 9.1 with kernel 2.4.21-013mdk and Linux Red Hat 9 with kernel 2.4.22. **Windows:** Windows XP Professional 2002, Service Pack 1.
Experiment Duration	T = 60 s
Traffic Details	CBR, Constant Bit Rate
	Protocol: UDP
	C = packets per second (pps or pkt/s)
	c = Packet Size (byte)

FIGURE 2.15 – Ensemble des spécifications expérimentales

Le concept de noyau et de système d'exploitation sera décrit en détail dans le chapitre suivant. Aucune autre information n'est donnée sur le matériel. Plusieurs ordinateurs peuvent correspondre à ces caractéristiques. Il n'est pas possible de réitérer les expériences sans identifier avec précision le matériel utilisé.

2.3.2.3 Analyse comparative

Nous présentons maintenant Les quatre tests de l'analyse comparative réalisés par les auteurs de D-ITG. Elle compare les performances de six générateurs : D-ITG, MTools, RUDE/-CRUDE, MGen, Iperf et UDPGen. Les deux premiers tests concernent Linux. Les deux derniers sont relatifs à Windows. Pour chaque système d'exploitation, le premier test expose les résultats d'une collecte d'informations auprès de l'émetteur uniquement. Le second illustre une collecte

7. Illustration issue de [12]

auprès de l'émetteur et du récepteur. Nous avons trouvé ces tests illustrés sur plusieurs support (voir[12], [20] et [21]). Cependant, nous n'avons trouvé aucune justification ou information sur le choix des valeurs de PS et d'IDT utilisées. Pour chaque test, nous donnons la description des paramètres de l'expérience. Ensuite nous présenterons les graphiques du cahier des charges. Pour le test 4 relatif à Windows XP, il est stipulé dans le cahier des charges la phrase suivante : *" Nous pouvons analyser uniquement D-ITG car il n'y a pas d'autres générateurs de trafic sur les plateformes Windows capable de collecter de l'information auprès du récepteur et de l'émetteur ".*

Test 1 : Linux, collecte d'informations coté récepteur

- IDT : 77 000 paquets par seconde donc $IDT = \frac{1}{77000} \approx 1,30 \cdot 10^{-5} s$
- PS : 1024 bytes
- Débit théorique : $8 \cdot 1024 \cdot 77000 = 630,784\ Mb/s$
- Durée : 60 s

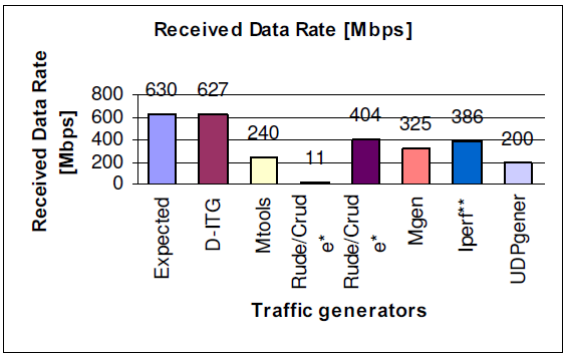

FIGURE 2.16 – Test 1 : Linux, collecte d'informations coté récepteur

Test 2 : Linux, collecte d'informations coté émetteur et récepteur

- IDT : 75 000 paquets par seconde donc $IDT = \frac{1}{75000} \approx 1,33 \cdot 10^{-5} s$
- PS : 1024 bytes
- Débit théorique : $8 \cdot 1024 \cdot 75000 = 614,4\ Mb/s$
- Durée : 60 s

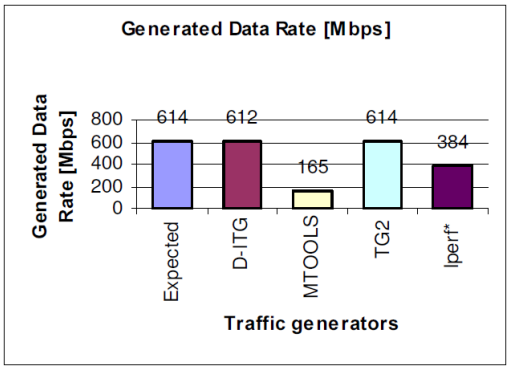

FIGURE 2.17 – Test 2 : Linux, collecte d'informations coté émetteur

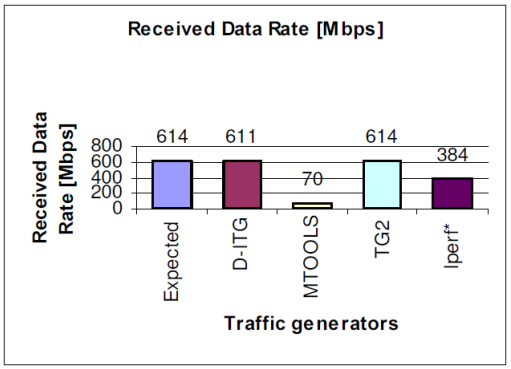

FIGURE 2.18 – Test 2 : Linux, collecte d'informations coté récepteur

Test 3 : Windows XP, collecte d'informations coté récepteur

- IDT : 30 000 paquets par seconde donc $IDT = \frac{1}{30000} \approx 3,33 \cdot 10^{-5} s$
- PS : 1024 bytes
- Débit théorique : $8 \cdot 1024 \cdot 30000 = 245,76\ Mb/s$
- Durée : 60 s

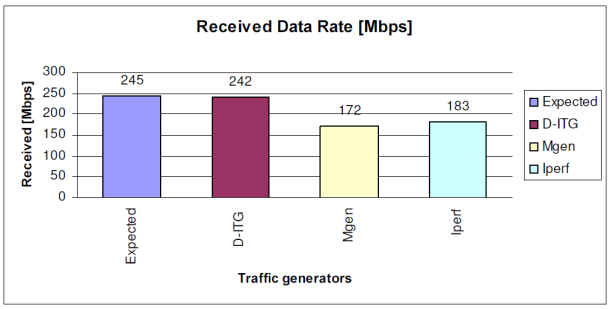

FIGURE 2.19 – Test 3 : Windows XP, collecte d'informations coté récepteur

Test 4 : Windows XP, collecte d'informations coté émetteur et récepteur

- IDT : 77 000 paquets par seconde donc $IDT = \frac{1}{77000} \approx 1,29 \cdot 10^{-5} s$
- PS : 1024 bytes
- Débit théorique : $8 \cdot 1024 \cdot 30000 = 245,76\ Mb/s$
- Durée : 60 s

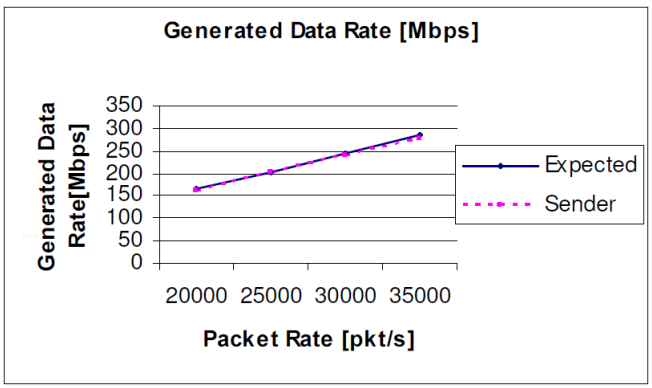

FIGURE 2.20 – Test 4 : Windows XP, collecte d'informations coté émetteur

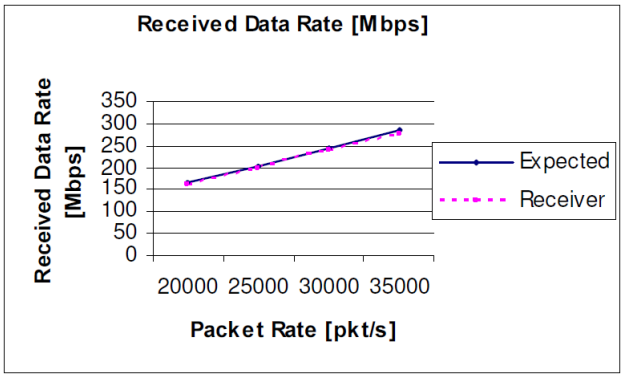

FIGURE 2.21 – Test 4 : Windows XP, collecte d'informations coté récepteur

2.3.2.4 Analyse des performances

Nous présentons dans cette section l'analyse des performances évoquée dans la cahier des charges. Elle concerne D-ITG uniquement. Précisons d'abord le vocabulaire. Un scénario distribué est un échange de donnée entre deux ordinateurs. Cette situation est celle évoquée à la sous-section 2.3.2.2 page 25. Un scénario local désigne une exécution de ITGSend et ITGRecv sur un même ordinateur. C'est une situation possible. La figure 2.13 page 23 illustre le cas où un ordinateur joue le rôle d'émetteur et de récepteur simultanément. La figure 2.23 montre le débit maximal obtenu dans le cadre d'un scénario distribué. La figure 2.22 montre le débit maximal obtenu dans un cadre d'un scénario local. Lorsque le symbole " $\geq$ " est placé devant la valeur du débit, la valeur indiquée est une borne inférieur au débit du trafic réellement généré. Les deux illustrations sont issues de [12]

Scenarios	Sender side data rate	Receiver side data rate
Linux OS—Both sender and receiver log	511 Mbit/s	≥511 Mbit/s
Linux OS—The receiver only logs	≥611 Mbit/s	≥611 Mbit/s
Window OS—Both sender and receiver log	102 Mbit/s	≥102 Mbit/s
Windows OS—The receiver only logs	≥241 Mbit/s	≥241 Mbit/s

FIGURE 2.22 – D-ITG – Débit maximal dans un scénario local

Scenarios	Sender side data rate	Receiver side data rate
Two workstations with Linux OS—Both sender and receiver log	612 Mbit/s	≥612 Mbit/s
Two workstations with Linux OS—The receiver only logs	≥627 Mbit/s	≥627 Mbit/s
Two workstations with Windows OS—Both sender and receiver log	161 Mbit/s	≥161 Mbit/s
Two workstations with Windows OS—The receiver only logs	≥242 Mbit/s	≥242 Mbit/s
One workstation with Linux OS and the other with Windows OS—The receiver only logs	≥627 Mbit/s (Linux)	483 Mbit/s (Windows)
One workstation with Linux OS and the other with Windows OS—The receiver only logs	242 Mbit/s (Windows)	≥627 Mbit/s (Linux)

FIGURE 2.23 – D-ITG – Débit maximal dans un scénario distribué

2.4 Discussion

Le cahier des charges présentait les fonctionnalités de D-ITG et ses performances. La description des fonctionnalités est conforme à l'application. La partie expérimentale nous permet de faire trois observations majeures sur les performances d'un générateur de trafic.

Premièrement, les résultats expérimentaux doivent être nuancés. Ils n'illustrent pas une performance supérieure de D-ITG. Les explications données sur ces tests ne sont pas exhaustives. D'autres articles présentent des comparaisons de générateurs de trafic. Ainsi, dans [60], les auteurs comparent la performance de quatre générateurs de trafic. Ces générateurs sont : "Iperf", "Netperf", "D-ITG" et "IP Traffic". L'expérience est la suivante : *" Pour des tailles de paquets différentes (i.e PS) variant de 128 Bytes à 1408 Bytes, le trafic TCP circulant sur le lien avec chacun des quatre générateurs "*. L'article présente un montage expérimental similaire à [19] : *" Deux ordinateurs équipés du système d'exploitation Windows connectés via un lien d'un débit de 100 Mb/s "*. Les résultats de l'expérience sont synthétisés dans un graphique illustré à la figure 2.24 [8]. Ce graphique donne une autre illustration des performances de D-ITG. Il est également disponible dans une autre étude comparative des générateurs de trafic (voir [75]). Dans cet article, Sudhakar Mishra, Shefali Sonavane et Anil Gupta montrent qu'il n'existe pas de générateur meilleur que tous les autres. Les performances sont nuancées et varient selon les fonctionnalités de ces outils.

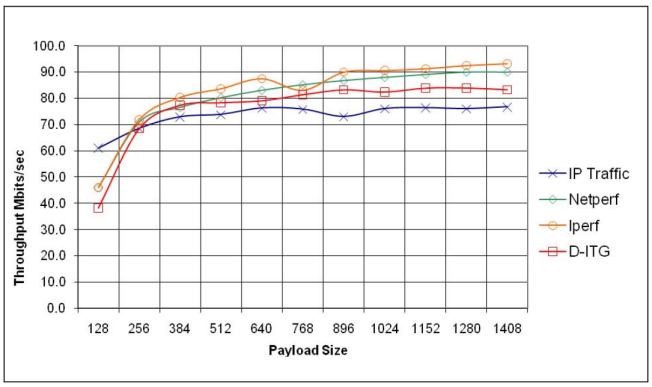

FIGURE 2.24 – Performances de Iperf, Netperf, D-ITG et IP Traffic avec le protocole TCP

Pour le cahier des charges et l'article disponible en [60], la description du matériel utilisé est trop sommaire. Elle ne permet pas de reproduire l'expérience. Il manque des informations essentielles. En effet en 1997 déjà, dans [86], Paxson évoquait les limites de performance induites par les transferts de données entre les composants d'un ordinateur pour la génération de trafic. La rapidité des transferts de données influence la performance des applications. Plus les transferts sont rapides, plus les performances sont élevées. La rapidité dépend spécifiquement de la mémoire utilisée et des supports de communications entre ces mémoires. En informatique, un tel support de communication s'appelle un bus. Un ordinateur en contient plusieurs. Ils sont placés sur la carte mère et relient les composants entre eux. La carte mère est le circuit imprimé principal de l'ordinateur sur lequel viennent se greffer l'ensemble des autres composants. Dans [19] et [60], seule la quantité de mémoire centrale est expliquée. Aucune information ne permet

8. Illustration issue de [60]

de déterminer le type et la norme de la mémoire employée. Aucune information ne permet d'identifier la carte mère utilisée. Or, c'est là le premier facteur d'influence sur les performances des générateurs. Les capacités de traitement d'un ordinateur dépendent de la rapidité de ces transferts. Ainsi, changer un ordinateur par un autre plus récent permet d'avoir de meilleures performances. C'est typiquement la raison pour laquelle nous achetons un nouvel ordinateur. Disposer de plus grandes capacités de traitement augmente la performance des générateurs de trafic.

Deuxièmement, l'analyse comparative a montré des différences de performance entre les générateurs. Sur le terrain, ces variations de performance peuvent avoir des conséquences. Les résultats expérimentaux pourraient être invalidés. Prenons un cas de figure. Soit une série de N expériences à réaliser. Chaque expérience requiert de générer un trafic réseau identique. Pour assurer une génération de trafic identique entre les tests, nous utilisons le même générateur de trafic sur l'ensemble de ces expériences. Un changement de générateur pourrait provoquer une modification du trafic généré. La lecture du cahier des charges ne suffit pas. Nous devons estimer les performances d'un générateur dans le contexte de l'expérience. Tout dépend des fonctionnalités utilisées. Dans la pratique, nous pourrions vouloir changer de générateur pendant nos expériences. Par exemple, utiliser la nouvelle version récemment parue du générateur. Or, les performances de l'application peuvent avoir changé. Il faut déterminer si le trafic généré par ces outils autorise la substitution d'un générateur par un autre. C'est rendu possible par un test préalable aux expériences à réaliser. Il ne faut pas faire l'hypothèse d'une performance uniforme entre les générateurs ; en ce compris les différentes versions d'une même application. Ils doivent être testés dans le contexte des expériences à réaliser. La substitution directe d'un générateur par un autre n'est pas sans risque pour la validité des résultats expérimentaux.

Troisièmement, l'analyse des performances de D-ITG à mis en évidence des différences de performances selon le système d'exploitation utilisé. Linux et Windows n'offrent pas les mêmes débits. Dès lors, effectuer une expérience avec D-ITG en faisant varier le système d'exploitation implique des changements dans les résultats expérimentaux. Typiquement, deux scientifiques effectuant les mêmes tests sur leurs machines personnelles avec un même générateur de trafic n'obtiendront pas systématiquement des résultats similaires. Ainsi, il ne faut pas considérer que les performances d'un générateur. Il faut considérer le couple formé par le générateur de trafic et le système d'exploitation. En conséquence, nous pouvons identifier trois facteurs influençant les performances d'une génération de trafic :

— Les capacités de traitement du matériel informatique
— Le choix du système d'exploitation
— Le choix du générateur de trafic

Les capacités de traitement et le choix du système d'exploitation influencent également les performances du générateur. Notre expérience présentée au chapitre 5 page 43 est en continuité des faits exposés dans la littérature. Nous chercherons à déterminer dans quelle mesure des modifications au sein d'un système d'exploitation peuvent affecter les performances de D-ITG. Pour cela, nous avons besoin d'un système d'exploitation. Dans le chapitre suivant nous continuerons notre analyse descendante. Nous quitterons les applications et nous passerons au système d'exploitation. Nous verrons quels sont les mécanismes qui influencent les performances des générateurs. Nous sélectionnerons aussi un système d'exploitation représentatif pour notre expérience comme nous l'avons fait avec les générateurs de trafic.

Chapitre 3

Systèmes d'exploitation

L'objectif de ce chapitre est de choisir un système d'exploitation pour notre expérience. Nous avions donné une définition générale de ce concept à la sous-section 2.1.3.2 page 11. Dans ce chapitre, nous allons approfondir cette notion. Un système d'exploitation est une application complexe et structurée. Elle est composée d'un ensemble de logiciels spécialisés et de fichiers. Un fichier se définit comme une collection organisée d'informations regroupées sous un même nom. Pour un rappel sur les définition de logiciel et d'application, consulter la section 2.1 page 3. Le système d'exploitation permet l'utilisation des ressources matérielles d'un ou plusieurs systèmes informatiques. Un système informatique peut être : un ordinateur, un smartphone, une tablette, un pocket PC, etc. Il assure également le démarrage du système et l'exécution des applications. Il exerce deux fonctions majeures. Premièrement, la gestion des ressources matérielles du système. Deuxièmement, il offre un ensemble de services et d'opérations au travers de plusieurs interfaces. Nous avions expliqué le concept d'interface de programmation à la sous-section 2.1.3.3 page 13. Ce chapitre présentera le concept plus général d'interface. Les systèmes d'exploitations modernes supportent plusieurs applications concurrentes et plusieurs utilisateurs simultanés. Ils doivent assurer l'indépendance de l'utilisateur et des applications vis-à-vis des composants physiques de l'ordinateur.

Dans le cadre de notre expérience, nous devons choisir un système d'exploitation nous permettant de modifier ses composants internes. Pour cela, nous exposerons dans la section 3.1 les principes généraux relatifs à la structure des systèmes d'exploitation. La section 3.2 page 34 évoque les mécanismes mis en oeuvre pour assurer la gestion des ressources. Enfin, la section 3.3 page 36 présentera le système d'exploitation retenu et donnera la justification de ce choix.

3.1 Structure des systèmes d'exploitation

Partons d'un ordinateur usuel. Lorsqu'il est en état de marche, la première chose que nous voyons est l'écran principal du système d'exploitation. Il s'agit du "bureau". C'est la première interface offerte par le système d'exploitation. Une interface définit la frontière de communication entre deux entités. Cette interface assure la gestion des interactions avec l'utilisateur. Elle permet à l'utilisateur d'interagir avec l'ordinateur. L'interface offre des opérations pour effectuer des tâches de base : manipulation de fichiers et lancement d'applications. Le lancement d'une application signifie l'exécution de cette application par ordre explicite de l'utilisateur. Par exemple, un double clic sur l'icône de l'application.

Une application s'exécute conventionnellement dans l'espace utilisateur. Cet espace est contrôlé par le système d'exploitation pour des raisons de sécurité et de robustesse. La sous-section 2.1.3.3 page 13 présentait le concept de bibliothèque. Les bibliothèques proviennent soit d'une source extérieur, soit du système d'exploitation. Ainsi, une application contient deux types d'opérations. D'une part, ses propres opérations regroupées sous le nom de code applicatif. D'autre part, les opérations fournies par les bibliothèques et utilisées dans le code applicatif. Le code applicatif et les bibliothèques externes font appel à la bibliothèque standard du système d'exploitation. Par exemple, Linux dispose de la *" GNU C Library "* abrégé glibc. Cette bibliothèque fournit un ensemble de services standardisés aux applications (voir [48]). Elle répond aux normes POSIX (voir [142]). La bibliothèque standard masque la diversité des programmes sous-jacents. Elle présente la vision d'une *" machine virtuelle "*, fournissant un ensemble d'opérations de base pour l'écriture de programmes. Un appel réalisé sur une de ces opérations de base se nomme un appel système. C'est la frontière entre l'espace utilisateur et l'espace physique.

Un appel système regroupe un ou plusieurs appels aux opérations du noyau. Le noyau contient le code fondamental d'un système d'exploitation. Il interagit directement avec les composants physiques du système. Ainsi, le noyau et les appels systèmes peuvent varier d'un ordinateur à l'autre. Ils dépendent de l'architecture des ordinateurs. L'architecture définit les composants physiques et l'ensemble des instructions supportées par un système. Il contient plusieurs millions de lignes de code. Par exemple, on estime que le noyau Linux contient approximativement 11,5 millions de lignes de code (voir [41]). Le noyau permet d'effectuer les tâches fondamentales :

— Lancement, exécution et terminaison des applications, logiciels et programmes
— Gestion et accès aux fichiers
— Gestion des utilisateurs
— Gestion des ressources
— Gestion de l'énergie

Le noyau est placé dans l'espace noyau. C'est un espace privilégié. Le noyau peut accéder à toutes les ressources du système sans limitations. en revanche, les applications de l'espace utilisateur ne peuvent utiliser que les ressources qui leurs sont allouées par le noyau. En effet, les applications sont en concurrence les unes par rapport aux autres pour accéder aux ressources du système. Elles peuvent accéder aux ressources seulement par des appels systèmes. Une ressource correspond à un composant physique ou logiciel susceptible d'être utilisé par une application. Tout accès à celles-ci est géré par le noyau. Il assure la gestion de la concurrence par des mécanismes internes. Le composant du noyau responsable du partage des ressources entre les applications s'appelle ordonnanceur. Nous exposerons ces mécanismes et leur fonctionnement dans la section 3.2 page 34. Il est nécessaire d'avoir une vue d'ensemble du système d'exploitation pour les comprendre. Nous avions évoqué les modules dans la sous-section 2.1.3.4 page 14. Le noyau utilise des modules. Ce sont des programmes permettant de lui ajouter des fonctionnalités. Un module peut être chargé à la demande dans l'espace noyau. Il est considéré comme un composant du système d'exploitation et bénéficie d'un accès privilégié aux ressources du système.

Un système est un ensemble de composants matériels. Pour gérer l'ensemble de ces composants, le noyau repose sur des pilotes. Un pilote est aussi nommé driver ou pilote de périphérique. C'est une interface logicielle permettant au noyau d'utiliser un périphérique. Comme il existe un très grand nombre de composants, les pilotes offrent une interface de programmation standard et conforme aux exigences du noyau. Ils masquent les spécificités propres aux périphériques. Les drivers standardisent les échanges entre le noyau et les composants physiques du système. Ils représentent la frontière entre l'espace noyau et l'espace physique. Chaque périphérique est équipé d'un contrôleur. C'est une puce électronique spécialisée dans les échanges de données. Elle est responsable de la gestion du périphérique. Le contrôleur interagit avec le driver pour dialoguer avec le noyau. Le couple formé par le driver et le contrôleur définit la charnière entre la partie logicielle et matérielle du système. Un contrôleur peut gérer un ou plusieurs périphériques. La figure 3.1 illustre la structure générale d'un système d'exploitation et d'une application. On y retrouve la notion d'espace évoqué à la sous-section 2.1.3.2 page 11.

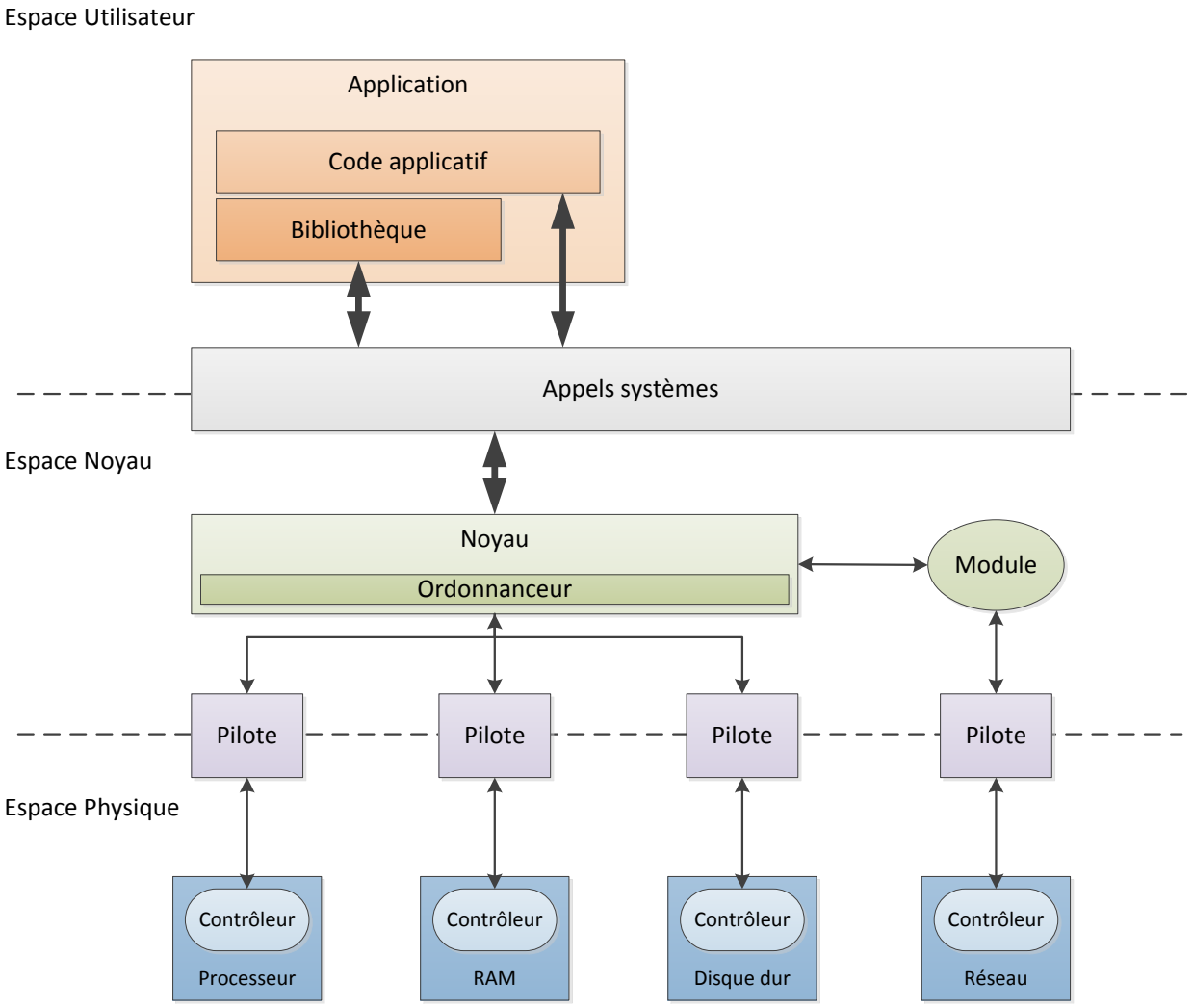

FIGURE 3.1 – Vue conceptuelle – Structure d'un système d'exploitation

3.2 Gestion des ressources

Prenons un cas de figure. Soit une usine spécialisée dans le montage de machines mécaniques. Pour maximiser la rentabilité, elle fonctionne 24h/24. Le chef d'atelier dispose de trois équipes de travail. Chacune des équipes travaille durant huit heures. Les horaires se composent comme suit :

— Équipe 1 : 8h00 à 16h00
— Équipe 2 : 16h00 à 0h00
— Équipe 3 : 0h00 à 8h00

Dans l'usine se trouve une horloge reliée à une alarme. Elle retentit périodiquement toutes les huit heures. Soit aux heures suivantes : 8h00, 16h00 et 0h00. L'alarme est un signal périodique qui interrompt les travailleurs. Un retentissement correspond à un arrêt de travail pour une équipe et au début de la journée pour une autre. Ainsi, l'usine fonctionne à 100 % de ses capacités. Si une équipe arrive à la fin du travail de montage avant la fin de ses huit heures alors le chef d'atelier est prévenu et une nouvelle tâche leur est confiée.

Un ordinateur fonctionne suivant ce cas de figure. Les logiciels sont comme des équipes de travail. Ils s'exécutent les uns à la suite des autres sur le processeur. Lors du démarrage de l'ordinateur, le système d'exploitation programme l'horloge matérielle pour émettre une interruption périodique. La périodicité s'exprime en Hertz. Elle indique le nombre d'interruptions par seconde. Les architectures actuelles supportent des valeurs allant de 100 Hertz à 1000 Hertz. Dans les systèmes d'exploitation généralistes tels que Windows, GNU/Linux et Mac OS, cette valeur est fixe. Elle ne peut être changée. Les interruptions de l'horloge matérielle permettent de contrôler quand et combien de temps exécuter des programmes. Une interruption de l'horloge matérielle se nomme un *"tic d'horloge"*.

Dans la pratique, on associe à chaque programme un compteur. Ce compteur indique le nombre de tic d'horloge pendant lequel le programme peut s'exécuter sur le processeur. A chaque tic, ce compteur est décrémenté d'une unité. Lorsqu'il atteint zéro, le programme a atteint la fin de sa durée d'accès au processeur. Un autre programme prend sa place. Dès lors, il existe deux possibilités. Premièrement, le programme a terminé son exécution. Le système d'exploitation clôture le programme et libère les ressources occupées par celui-ci. Deuxièmement, le programme doit poursuivre son exécution. Dans ce cas, le compteur du programme est réinitialisé à sa valeur maximale. Il doit attendre pour effectuer un prochain passage sur le processeur : il est placé dans la listes des programmes éligibles pour le processeur. Un programme bénéficie d'une option supplémentaire. Il peut demander explicitement à être suspendu de son accès au processeur pour une durée précisée. Si la durée d'attente est strictement supérieure à zéro, elle est décrémentée à chaque tic d'horloge. Lorsqu'elle atteint zéro, le programme est replacé dans la liste des programmes éligibles pour l'accès au processeur. Sinon, le programme est directement placé dans cette liste.

Le noyau est le composant du système d'exploitation qui assure la gestion des interruptions. C'est l'ensemble des mécanismes de gestion des interruptions qui définissent les priorités d'accès

aux ressources de l'ordinateur. Nous avons décrit un cas de figure dans lequel les programmes s'exécutent seuls sans interaction avec l'utilisateur et les périphériques de l'ordinateur. Dans la pratique, la gestion des interruptions est plus complexe. Une interruption peut interrompre l'exécution d'un programme. Prenons un exemple : l'utilisateur enfonce une touche du son clavier. Une interruption est un signal électrique émis par le contrôleur d'un périphérique. En l'occurrence, le contrôleur du clavier. Les interruptions sont adressées au contrôleur d'interruption. C'est une puce électronique possédant plusieurs entrées pour les interruptions et une sortie connectée au processeur. Le rôle du gestionnaire d'interruption est de gérer la concurrence entre ces interruptions. Il les présente séquentiellement par ordre d'importance au processeur qui ne peut en traiter qu'une seule à la fois. Lorsque le processeur détecte une interruption, il interrompt le programme en cours d'exécution pour traiter cette interruption. Le processeur notifie le noyau qui peut gérer cette interruption de manière appropriée.

La gestion d'une interruption se déroule comme suit. A chaque périphérique est assigné une valeur numérique unique d'interruption. Ces valeurs sont nommées IRQ pour "Interrupt ReQuest". Historiquement, l'IRQ 0 était conventionnellement associée à l'horloge matérielle. Maintenant, les IRQ sont associées aux périphériques dès le démarrage de l'ordinateur par le système d'exploitation. L'objectif est d'assigner une valeur IRQ unique à chaque périphérique. Ces valeurs permettent au noyau d'identifier la routine d'interruption à invoquer pour traiter l'interruption en cours. Une routine d'interruption est une simple fonction écrite en langage C. Elle est fournie par le pilote du périphérique dont émane l'interruption. c'est elle qui réalise les opérations nécessaire pour traiter l'interruption. Dans notre exemple, la routine pourrait déterminer la touche enfoncée par l'utilisateur et afficher le symbole correspondant sur l'écran. Les routines d'interruption doivent correspondre à un prototype définit par le noyau. Le prototype assure la compatibilité des échanges de données entre les routines et le code du noyau. Puisque les interruptions peuvent se produire à tout instant, les routines d'interruptions peuvent être invoquées à tout moment. Elles doivent être aussi brèves que possible. Au minimum, elles doivent acquitter les interruptions. La plupart d'entre elles associent des instructions de traitement à l'acquittement. Quand le noyau a invoqué une routine d'interruption, il se trouve dans un mode d'exécution spécifique. Il ne peut être interrompu par une autre interruption. C'est un mode d'exécution différent de celui des programmes. Une routine ne peut pas être interrompue ou bloquée. Elle doit s'exécuter le plus rapidement possible car elle bloque l'exécution d'un programme.

Le composant du noyau qui est responsable de la gestion des interruptions et de l'exécution des programmes est l'ordonnanceur. Il choisit comment les programmes sont exécutés sur le processeur. Plusieurs algorithmes existent (voir [117]). Par exemple, la méthode du tourniquet ou "Round-robin" qui attribue une durée fixe d'exécution à chaque programme. L'ordonnanceur définit comment les interruptions sont gérées. Changer les choix de gestion des interruptions influencent la performance des applications. Il existe deux grande familles d'ordonnanceurs : d'une part les ordonnanceurs généralistes ; d'autre part les ordonnanceurs temps réel. Ils sont utilisés dans des domaines d'application différents. Un systèmes d'exploitation équipé d'un ordonnanceur temps réel est appelé système d'exploitation temps réel. Un système d'exploitation équipé d'un ordonnanceur généraliste est appelé système d'exploitation généraliste. Nous détaillons ces deux catégories de systèmes d'exploitation dans les sous-sections 3.2.1 et 3.2.2 page 36.

3.2.1 Systèmes d'exploitation généralistes

Les systèmes d'exploitation généralistes sont les plus répandus. Ils correspondent aux systèmes pré-installés sur les ordinateurs à destination des marchés de masse. Ces systèmes d'exploitation conviennent pour des tâches quotidienne : bureautique, internet et messagerie, jeux et multimédias. Ce sont les plus utilisés actuellement. Les objectifs de ces systèmes sont multiples : ergonomie, robustesse et compatibilité avec les applications. Les systèmes d'exploitation les plus connus sont :

— Microsoft Windows (voir [148])
— Mac OS et IOS (voir [147])
— GNU/Linux (voir [144])
— Google chrome OS (voir [89])

3.2.2 Systèmes d'exploitation temps réel

L'objectif d'un système d'exploitation temps réel est de pouvoir réagir très rapidement à un événement et dans un délai de temps certifié. Il subit des contraintes de temps fortes. Par exemple, une application de vidéoconférence nécessite la synchronisation entre les flux audio et vidéo. Les applications pratiques sont nombreuses. Elles sont toutes dites "temps critique". Ainsi, on retrouve des noyaux temps réel dans les chaînes de montage des industries, les dispositifs GPS, les simulateurs de vols voir encore les ordinateurs de bord des navettes spatiales (voir [114]). Les systèmes d'exploitation temps réel les plus connus sont :

— Intel IRMX (voir [102])
— Projet Linux RT (voir [44])
— Microsoft Windows avec l'application RTX IntervalZero (voir [1])

3.3 Choix d'un système d'exploitation

Nous avons besoin d'un système d'exploitation dont les composants peuvent être modifiés. Nous souhaitons comparer les performance de D-ITG sur un même système d'exploitation équipé d'un noyau généraliste et d'un noyau temps réel. L'utilisation d'un noyau temps réel se justifie aux regards de la littérature scientifique. De tels noyaux ont déjà été étudiés dans le cadre des applications multimédias par Marcos Paredes-Farrera, Martin Fleury et Mohammed Ghanbari (voir [85]). L'étude des noyaux temps réel avec les générateurs de trafic prend naissance dans la simulation et l'émulation du trafic issu des jeux vidéos en ligne et des applications multimédias. Ces noyaux permettent des latences plus faibles. C'est une qualité appréciable pour ces deux domaines. Nous avons trouvé un autre projet de recherche ayant pour objectif de concilier les noyaux temps réel avec les besoins des générateurs. Il s'agit du projet Kurt de l'université du Kansas. Le projet se présente comme un patch à appliquer au code source d'un noyau Linux en version 2.4 ou 2.6. Il offre une interface de programmation aux développeurs d'applications temps critique. Dans [113], l'auteur évoque le développement d'un générateur logiciel basé sur Kurt. Ce type de développement cause un problème. Il rend le générateur de

trafic dépendant d'une version spécifique du noyau. C'est une contrainte plus liante que la dépendance au système d'exploitation. En effet, un système d'exploitation peut changer de noyau. Kurt nous oblige à maintenir ce noyau statique. Le support du noyau Linux 2.6 a officiellement pris fin en août 2011 après un cycle de vie de huit ans. Nous sommes actuellement à la version 4.1 du noyau. Les projets basés sur Kurt font maintenant partie de l'héritage technologique. Le projet Kurt semble avoir pris fin en même temps que la version 2.6 du noyau en 2011. Nous n'avons pas trouvé d'article plus récent sur l'étude des générateurs de trafic avec les systèmes d'exploitation temps réel. Or, les générateurs de trafic sont à considérer comme des applications temps réel. Prenons un cas précis : nous émettons un débit constant de N bit par seconde. En d'autres mots, nous avons la durée d'une seconde pour émettre ces N bits. Si nous doublons le volume de bits par seconde (2N) alors nous avons deux fois moins de temps pour envoyer un volume de données N. Ne pas respecter ce délai de temps en le dépassant implique de ne pas respecter le débit constant imposé. Plus le débit est élevé, plus le laps de temps disponible pour envoyer les données est court. C'est un des concepts des applications temps critique.

Nous avons envisagé quatre systèmes d'exploitation. Premièrement, Microsoft Windows 7. De base, ce système d'exploitation est généraliste. Lorsqu'il est doté de l'application RTX commercialisée par la société IntervalZero, Windows est doté de fonctionnalités temps réel (voir [1]). L'application RTX installe de multiples interfaces de programmation à plusieurs zones cibles du système d'exploitation. La figure 3.2 illustre l'architecture ainsi obtenue [1]. L'application RTX fonctionne comme le projet Kurt. Nous avons tenté de contacter l'entreprise mais nous n'avons jamais obtenu de réponse à nos sollicitations. Deuxièmement, le système d'exploitation Windows CE (voir [143]). Il est doté de capacité temps-réel mais à destination des systèmes embarqués et possédant des ressources limitées. Ce n'est pas un système d'exploitation généraliste. Nous ne l'avons pas utilisé pour cette raison. Troisièmement, nous avons analysé la gamme de produit de la société Apple. Nous n'avons trouvé aucune solution temps-réel.

Quatrièmement, le système d'exploitation GNU/Linux. Ce système d'exploitation est abondamment étudié dans la littérature scientifique. Nous utiliserons la distribution Debian pour notre expérience. Le système d'exploitation Debian est disponible gratuitement et sous licence GNU/GPL. Son code source est ouvert et disponible sur le site internet du projet (voir [92]). La dernière version en date était la version 8.1.0 lors de notre analyse. Nous avons sélectionné Debian pour deux raisons principales. D'une part pour ses qualités générales, d'autre part pour ses qualités techniques. Premièrement, c'est un système d'exploitation moderne largement déployé sur les ordinateurs personnels et les serveurs. Il est représentatif des systèmes d'exploitation actuels. Debian est reconnu pour sa grande stabilité dans sa version "stable" [2]. C'est cette version que nous utiliserons. Deuxièmement, Debian est un choix techniquement intéressant. Ce système d'exploitation offre une documentation fournie de son code source et de ses fonctionnalités. De plus, il est compatible avec les noyaux Linux dits "vanilla". Un noyau "vanilla" désigne un noyau Linux n'ayant subit aucune modification pour le rendre compatible avec le reste du système d'exploitation. Le projet Debian considère le noyau comme un constituant modifiable du système d'exploitation. D'autres systèmes tels que Windows et Mac OS le considère comme un composant non remplaçable. Ainsi, Debian offre la possibilité à l'utilisateur de choisir le noyau à utiliser lors du démarrage de l'ordinateur. Nous avons utilisé cette spécificité pour comparer les

1. http://intervalzero.com/assets/WindowsRTXArchitecture.png
2. le système d'exploitation Debian se décompose en trois branches : "stable", "testing" et "unstable".

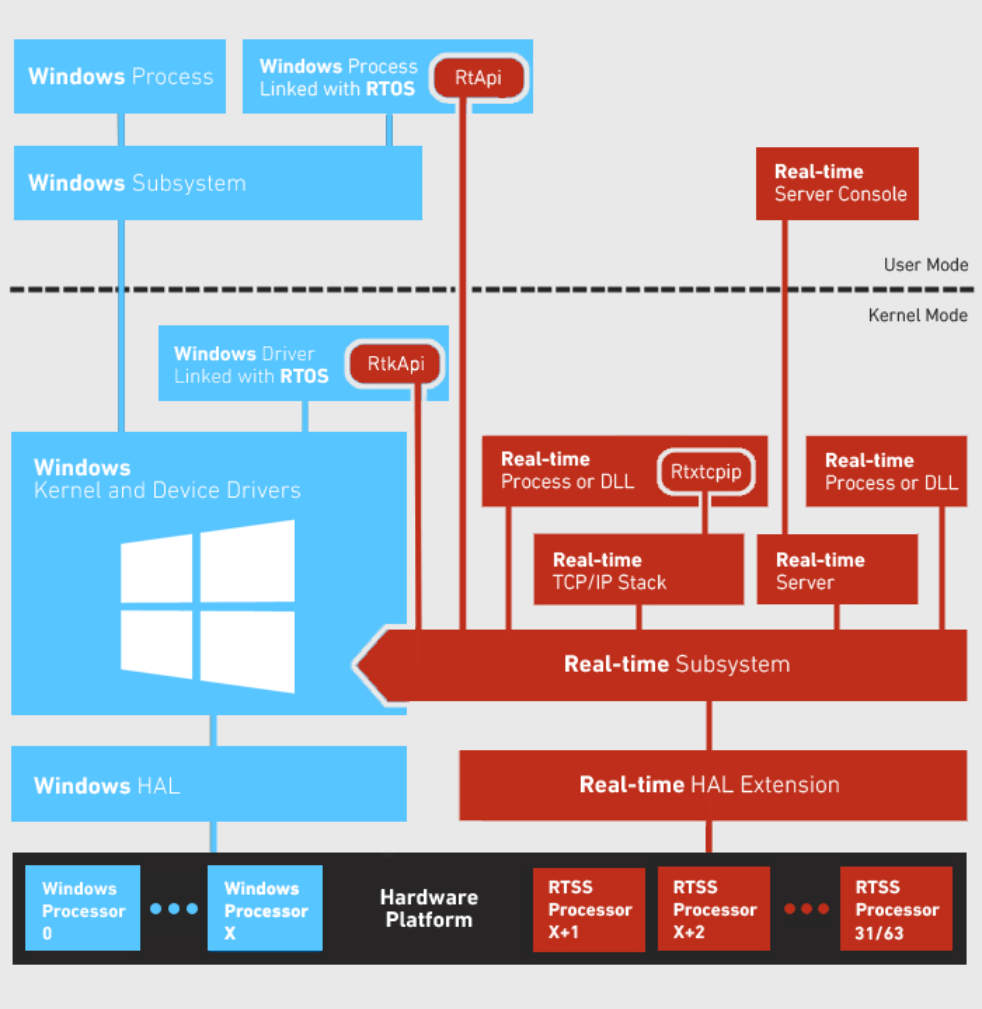

FIGURE 3.2 – Architecture de Windows (en bleu) adjoint de l'application RTX (en rouge)

performances d'un même générateur de trafic selon un noyau générique et un noyau temps réel. nous utiliserons le dernier noyau stable disponible lors de l'expérimentation (voir [41]). Il s'agit du noyau 4.0.5. Nativement, ce noyau est généraliste (voir [42]). Pour obtenir un noyau temps réel, on applique un ensemble de modifications sur le code source du noyau généraliste (voir [43]). Cet ensemble de modifications est regroupé dans un patch. Les modifications apportées sont relatives aux paramètres de préemption, de latence, de synchronisation, d'ordonnancement et de chronométrage. L'application du patch sur le code source est réalisé par un logiciel dédié manipulable en ligne de commande. Ce procédé rend le noyau temps réel obtenu compatible avec toutes les applications. Il ne fournit pas une interface de programmation comme le projet Kurt. Il offre la même bibliothèque standard qu'un noyau généraliste mais avec des capacités temps réel accrues.

Nous possédons maintenant un générateur de trafic et un système d'exploitation pour notre expérience. Nous continuerons notre analyse descendante dans le chapitre suivant. Nous sélectionnerons le réseau nécessaire à notre expérience.

Chapitre 4

Réseau et câblage

Ce chapitre à pour objectif de sélectionner l'architecture réseau de notre expérience. Nous avions défini le concept de réseau au chapitre 2 page 3 comme : *"Un ensemble d'entités matérielles et logicielles autonomes interconnectés les unes aux autres par des voies de communication"*. Nous présenterons d'abord les voies de communication à la section 4.1. Ensuite, les périphériques réseaux à la section 4.2 page 40. Enfin, la section 4.3 page 41 exposera le choix d'une architecture et la justification de ce choix.

4.1 Voies de communication

Une voie de communication désigne le support physique utilisé pour véhiculer les données entres les différentes entités du réseau. Il existe une grande diversité au sein de ces différents supports. La liste ici-bas présente quelques technologies. Elle est inspirée des informations disponibles dans [105].

- Réseau filaire :
  - Optique : fibre optique
  - Électrique : câble de cuivre à pairs torsadées, câble coaxial
- Réseau sans fil :
  - Optique : laser
  - Ondes électromagnétiques : maser, Wi-fi, WIMAX, liaisons mobiles et satellitaires

Entre les réseaux filaires et sans-fils, ce sont les réseaux filaires qui possèdent les meilleures performances. Il est plus facile de faire transiter des données dans un câble que dans les airs. Les réseaux sans fils subissent un ensemble de phénomènes susceptibles d'altérer leur performances. Ces phénomènes sont inhérents à l'environnement dans lequel est déployé le réseau sans fil :

- Réflexion
- Obstruction de la propagation
- Interférences

4.2 Périphériques réseaux

Il existe trois modes principaux de transfert de données dans les réseaux informatiques. Il s'agit des mode "simplex", "half-duplex" et "full duplex". Ces modes de transfert sont utilisé par les trois catégories de périphériques réseaux existantes. Il s'agit des "routeurs", "switches" et "hubs". Ces périphériques sont tous équipés de ports de connexion. Chaque port peut effectuer des entrée et des sorties de données. Nous présenterons d'abord les modes de transfert à la sous-section 4.2.1. Nous allons définir chacun d'eux et donner un exemple d'illustration sur base du trafic routier. Ensuite, nous expliquerons les catégories de périphériques à la sous-section 4.2.2. Nous exposerons leurs fonctionnalités et les modes de transfert utilisés.

4.2.1 Modes de transfert de données

Simplex :

Le mode simplex correspond à un mode de transmission de données unidirectionnel. Dans un réseau routier, il s'agit d'une route à sens unique. La figure 4.1 illustre cette situation.

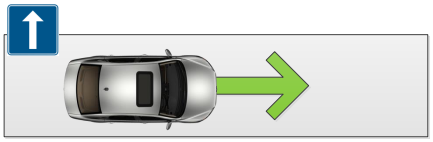

FIGURE 4.1 – Mode simplex – Une route à sens unique

Half-duplex :

Le mode half-duplex est un mode de transmission bidirectionnel. Le canal de communication utilisé est partagé entre les hôtes réseaux. Ils ne peuvent l'utiliser qu'un seul à la fois. Sinon, des collisions surviennent. C'est un mode de transmission simplex alterné. Dans un réseau routier, il s'agit d'une route étroite dont le trafic est régulé par des feux de signalisation. Les véhicules se partagent l'accès à la route en alternance. La figure 4.2 illustre cette situation.

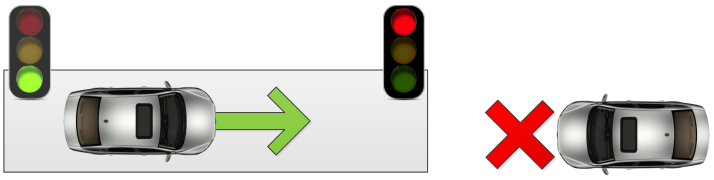

FIGURE 4.2 – Mode half-duplex – Une route avec feux de signalisation

Full duplex :

Le mode full duplex est un mode de transmission bidirectionnel. Le canal de communication utilisé peut être utilisé par plusieurs hôtes réseaux simultanément. En conséquence, le débit peut être doublement supérieur au mode de transmission half-duplex. Dans un réseau routier, il s'agit d'une route à deux voies de circulation ; l'une montante, l'autre descendante. La figure 4.3 illustre cette situation.

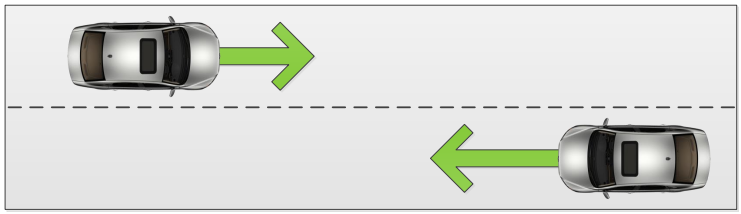

FIGURE 4.3 – Mode full duplex – Une route bidirectionnelle à deux bandes

4.2.2 Périphériques

Routeur :

Un routeur assure l'aiguillage des données entres des réseaux indépendants. On dit que le routeur assure le routage des données. Le routage est définit par un ensemble de règles contenue dans une table de routage. Une analogie possible est un échangeur d'autoroute. Selon les modèles, un routeur peut fonctionner en mode half-duplex ou full duplex. Le choix d'un mode est effectué par deux procédés. Soit le routeur sélectionne automatiquement un mode en observant les périphériques qui y sont connectés. Soit le mode est sélectionné manuellement.

Hub :

Un hub est un concentrateur de périphériques réseaux. Il permet à plusieurs hôtes réseaux de partager un même canal de communication. Lorsque des données se présentent sur un port, elles sont amplifiées puis envoyées sur l'ensemble des autres ports. Un hub ne fait que répéter les données qu'il reçoit. Ce fonctionnement implique nécessairement un mode half-duplex.

Switch :

Un switch agit comme un pont réseau. Lorsque des données se présentent sur un port, elles sont amplifiées puis envoyées uniquement sur le port concerné. Pour réaliser ce traitement, le switch conserve une table de correspondance entre les destinations des données et ses ports. Cette table est construite et maintenue à jour automatiquement par le périphérique. Un switch peut fonctionner en mode half-duplex et full duplex. Dans un réseau, un hub peut être remplacé par un switch.

4.3 Choix d'une architecture réseau

Nous souhaitons sélectionner une architecture réseau minimale et représentative des performances du réseau national belge. Nous avons consulté le rapport technique publié par la société Akamai active dans les réseaux et télécoms (voir [6]). Selon le premier rapport quadrimestriel de 2015, la vitesse moyenne du réseau belge est de 11,9 Mb/s avec des pics moyens de 53,5 Mb/s. Le réseau mobile obtient un débit moyen de 5,8 Mb/s dont 80% des connexions ont un débit inférieur à 4 Mb/s. Un réseau permet d'échanger des données entre des hôtes réseaux. Il y a donc au minimum deux hôtes sur un réseau. Les plus répandus sont les ordinateurs. Nous en utiliserons deux. Il nous faut encore les connecter l'un à l'autre. Pour obtenir un réseau similaire aux capacité du réseau belge, nous opterons pour un réseau câblé répondant à la dénomination *"Fast Ethernet"*. Il s'agit de l'ensemble des technologies mises en oeuvre permettant d'atteindre

un débit théorique maximal de 100 Mb/s. Un réseau câblé nous permettra d'isoler plus facilement notre montage expérimental de l'environnement qu'un réseau sans fils. Il subira moins de perturbations. Chaque ordinateur sera relié à un switch par un câble à base de pairs de cuivre torsadées. Ce sont les câbles les plus utilisés dans les entreprises et chez les particuliers. Ils sont utilisés depuis plus de trente ans maintenant. C'est une technologie très répandue, bon marché et accessible au plus grand nombre. Nous utiliserons un switch car il offre des fonctionnalités plus étendue qu'un hub. Son mode de fonctionnement permet d'atteindre des débits plus élevés. Nous n'utiliserons pas de routeur car son emploi n'est pas justifié compte tenu de notre montage. Un routeur permet de connecter des réseaux entre eux. Nous utiliserons un unique réseau minimaliste. Dès lors, l'emploi d'un routeur n'est pas adéquat. Ainsi, les ordinateurs seront connectés entre eux par l'intermédiaire d'un périphérique. Dans la pratique, c'est la situation la plus répandue. En effet, il est peu courant de trouver un ordinateur relié à un autre directement par un câble réseau. Ce type de montage correspond exclusivement à des besoins expérimentaux. La figure 4.4 présente un schéma de notre futur montage expérimental.

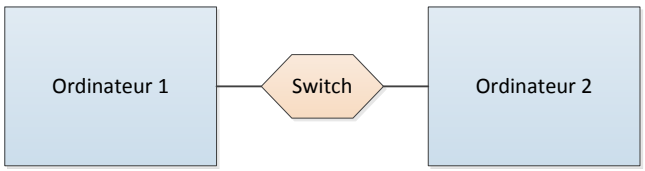

FIGURE 4.4 – Schéma de montage

Au chapitre 2 page 3, nous avons sélectionné le générateur de trafic D-ITG. Au chapitre 3 page 31, nous avons sélectionné le système d'exploitation Debian GNU/Linux. Dans le présent chapitre, nous avons sélectionné notre architecture réseau. Nous possédons tous les éléments nécessaires à la réalisation de notre expérience. Nous exposerons notre expérimentation dans le prochain chapitre. Nous exposerons nos résultats expérimentaux et tous les informations nécessaires à quiconque souhaiterait reproduire cette expérience.

Chapitre 5

Expérimentation

Nous avons sélectionné dans les chapitres 2, 3 et 4 les éléments nécessaires à notre expérience. Nous avons aussi expliqué le contexte et justifié nos choix. Nous allons maintenant procéder à la réalisation de l'expérience. Nous donnerons l'ensemble des informations nécessaire à sa reproduction. Nous présenterons notre expérience sous la forme d'un rapport de laboratoire. C'est la démarche expérimentale enseignée lors des travaux pratiques dispensés par la faculté des sciences de l'université de Namur. Cette démarche est composée de sept étapes. Nous exposerons chacune de ces étapes dans une section de ce chapitre :

1. Objectif de l'expérience : voir section 5.1 page 43
2. Protocole expérimental : voir section 5.2 page 43
3. Données et observations : voir section 5.3 page 54
4. Calculs : voir section 5.4 page 54
5. Résultats : voir section 5.5 page 56
6. Discussion : voir section 5.6 page 70

5.1 Objectif de l'expérience

L'objectif de l'expérience est d'analyser les caractéristiques du trafic généré par l'application D-ITG 2.8.1 avec un système d'exploitation GNU/LINUX Débian 8.0.1 équipé du noyau Linux généraliste 4.0.5 et du noyau temps-réel 4.0.5. Nous comparerons les trafics générés avec l'utilisation du noyau généraliste et du noyau temps-réel. Nous réaliserons notre analyse sur base d'un ensemble de métriques décrit dans la sous-section 5.2.3 page 47.

5.2 Protocole expérimental

Nous illustrons notre protocole expérimental en trois sous-sections. D'abord une présentation schématique du montage expérimental et la description complète du matériel utilisé à la section 5.2.1 page 44. Ensuite, la sous-section 5.2.2 page 45 présentera deux photographies du montage réalisé. Enfin, la sous-section 5.2.3 page 47 décrira notre manipulation. Nous y exposerons trois concepts : les mesures effectuées, le déroulement de l'expérience et les outils développés pour réaliser cette expérience.

5.2.1 Montage expérimental et matériel utilisé

Nous illustrons notre plan de montage par le schéma de la figure 5.1. L'ensemble du matériel utilisé est décrit dans la liste ci-dessous.

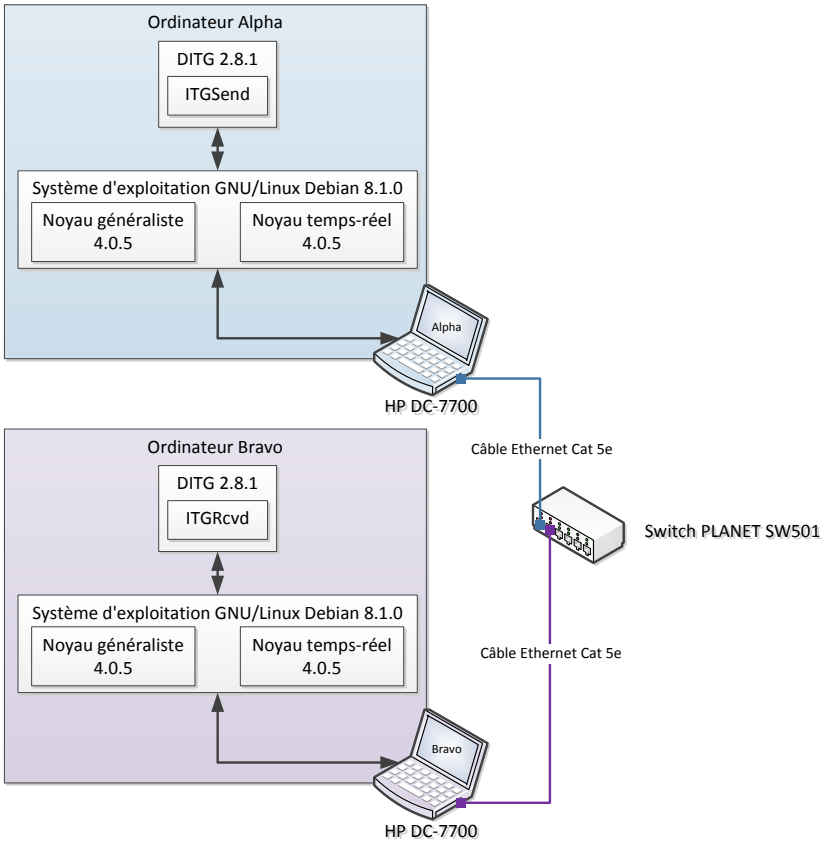

FIGURE 5.1 – Schéma de montage expérimental

Ordinateur :
Les ordinateurs intitulés Alpha et Bravo sont de marque HP et de modèle DC-7700. L'annexe D page 110 expose en détails les caractéristiques techniques de ces ordinateurs. Ils sont représentatifs du matériel standard moderne. Ce type de machine est utilisé dans la plupart des entreprises pour effectuer des tâches bureautiques usuelles. Il s'agit d'une configuration courante et largement répandue. L'emploi de deux ordinateurs identiques permet de faciliter l'analyse des résultats. Nous ne devons pas prendre en compte des capacités de traitement différentes entre les ordinateurs.

Switch :
Le switch utilisé est de marque "PLANET" et de modèle "SW-501". L'annexe E page 112 page 112 expose en détails les caractéristiques techniques de ce switch. Il est compatible avec la dénomination *"Fast Ethernet"* et permet d'atteindre des débits de 100 Mb/s.

D-ITG 2.8.1 :
Le logiciel D-ITG a été décrit dans le chapitre 2, à la sous-section 2.3.1 page 18.

GNU/Linux Debian 8.1.0 :
Le système d'exploitation Debian a été décrit dans le chapitre 3, à la sous-section 3.3 page 36.

Câble réseau :

Les câbles réseaux utilisés sont des câbles de catégorie 5E. Ils ont une longueur de deux mètres. Nous les avons sélectionnés car ils offrent un débit théorique de 1000 Mb/s. Ainsi, le débit des données transférées entre Alpha et Bravo ne peux pas être limité par la capacité de ces câbles réseaux.

5.2.2 Photographies du montage expérimental

Le schéma de la figure 5.1 page 44 et la description du matériel nécessaire permettent différentes interprétations sur le montage de cette expérience. Nous présentons deux photographies de notre montage dans cette sous-section. Une particularité est à signaler. Nous avons utilisé un commutateur écran-clavier-souris pour mutualiser le clavier, la souris et l'écran entre les ordinateurs Alpha et Bravo. L'utilisation d'un commutateur n'a aucun impact sur le processus d'expérimentation. Il permet de rendre le montage plus compact. Un troisième ordinateur intitulé "Charlie" est visible sur les illustrations. C'est un ordinateur de réserve identique à Alpha et Bravo. Il permet de répondre aux avaries potentielles du matériel. Les illustrations 5.2 et 5.3 page 46 sont des photographies du montage réalisé.

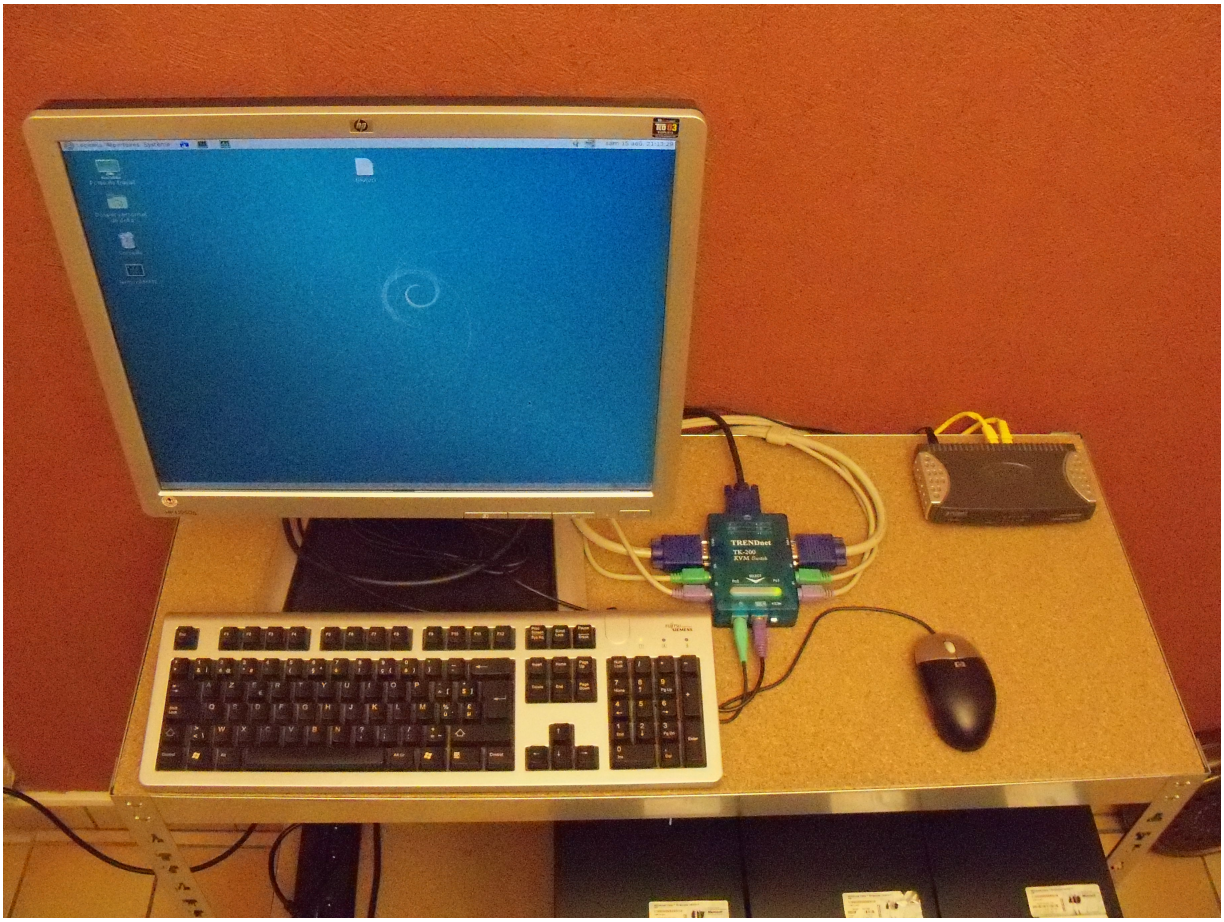

FIGURE 5.2 – Installation expérimentale vue plongeante

FIGURE 5.3 – Installation expérimentale vue de face

5.2.3 Description de la manipulation

Nous avons défini une série de douze tests à réaliser sur le noyau généraliste et le noyau temps-réel. Ces tests sont numérotés de 1 à 12. Pour chacun d'eux, nous utilisons le protocole UDP et des datagrammes de 100 bytes exactement. Nous utilisons UDP car il n'intègre pas de contrôle de congestion et de flux. Ce protocole introduit moins de paramètres dans l'analyse des résultats. Il a déjà été utilisé dans le cahier des charges de D-ITG lors de l'analyse comparative et l'analyse des performances (voir respectivement 2.3.2.3 page 25 et 2.3.2.4 page 28). Nous utilisons des datagrammes de 100 bytes car ils peuvent être encapsulés dans un seul paquet IP. Pour chaque test, le débit du trafic généré est constant. Nous nous sommes inspirés des articles [19] et [18] pour élaborer ces tests. La liste des tests est :

— Test 1 : 1 datagramme UDP de 100 bytes par seconde pendant 30 secondes
— Test 2 : 10 datagramme UDP de 100 bytes par seconde pendant 30 secondes
— Test 3 : 100 datagramme UDP de 100 bytes par seconde pendant 30 secondes
— Test 4 : 1000 datagramme UDP de 100 bytes par seconde pendant 30 secondes
— Test 5 : 10 000 datagramme UDP de 100 bytes par seconde pendant 30 secondes
— Test 6 : 100 000 datagramme UDP de 100 bytes par seconde pendant 30 secondes
— Test 7 : 125 000 datagramme UDP de 100 bytes par seconde pendant 30 secondes
— Test 8 : 187 500 datagramme UDP de 100 bytes par seconde pendant 30 secondes
— Test 9 : 250 000 datagramme UDP de 100 bytes par seconde pendant 30 secondes
— Test 10 : 312 500 datagramme UDP de 100 bytes par seconde pendant 30 secondes
— Test 11 : 375 000 datagramme UDP de 100 bytes par seconde pendant 30 secondes
— Test 12 : 3 750 000 datagramme UDP de 100 bytes par seconde pendant 30 secondes

Les tests 0 à 6 suivent une progression exponentielle en puissance de 10 pour atteindre le débit maximal théorique de 100 Mb/s. Les tests 7 à 11 spécifient un débit supérieur à 100 Mb/s. Nous souhaitons étudier le comportement du générateur de trafic dans des conditions limites. Pour chaque test, D-ITG nous permet d'obtenir un ensemble de mesures coté émetteur et récepteur. Ainsi, nous collecterons un ensemble de mesure sur Alpha et Bravo :

1. Le débit moyen en Kbit/s
2. Le délai "OWD" minimum, maximum et moyen (voir 2.3.1.1 page 18)
3. La gigue (voir 2.3.1.1 page 18)
4. Le nombre de paquets reçus et perdus
5. Débit instantané

Les mesures 1 à 4 s'expriment par rapport à la durée totale d'un test. C'est une durée égale à 30 secondes. La mesure 5 est obtenue tous les millièmes de seconde. C'est pourquoi nous parlons de mesure instantanée. Elle est effectuée sur un instant d'un millième de seconde. Chaque test sera réalisé dix fois pour éviter les faux positifs ou négatifs. Nous parlerons d'une occurrence de test pour désigner l'un de ces dix tests identiques. De plus, nous respecterons une période d'inactivité de 10 secondes entre chaque occurrence de test. Cela permet d'éviter les interférences entre chaque génération de trafic. Pour chaque test, nous réaliserons le calcul d'une moyenne arithmétique sur base des dix occurrences de ce test. Nous calculerons une

moyenne arithmétique pour toutes les mesures définies dans la liste précédente. Ensuite, pour chaque test, nous calculerons une moyenne mobile du débit instantané sur base de la moyenne arithmétique du débit instantané des occurrences d'un test. L'analyse de la moyenne mobile nous permettra de faire deux observations. D'une part, vérifier si le débit du trafic produit par D-ITG est conforme avec le débit spécifié. D'autre part, vérifier l'existence de différences de débit entre le noyau généraliste et le noyau temps-réel pour un même test. La section 5.4 page 54 explique les concepts de moyenne arithmétique et de moyenne mobile. Ainsi, les résultats générés à partir de ces tests seront statistiquement significatifs. La méthodologie de mesure est présentée ci-dessous. Elle est systématique et définie sous la forme d'un algorithme textuel :

1. Mise sous tension des ordinateurs Alpha et Bravo
2. Sélection d'un noyau pour lequel il n'existe pas encore de résultats expérimentaux
3. Vérification du bon fonctionnement général de Alpha et Bravo
4. Vérification du bon fonctionnement général du réseau
5. Est-ce que tous les tests ont étés réalisés ? Si oui aller à la ligne 16, sinon poursuivre
6. Prendre le premier test non réalisé dans l'ordre croissant de la liste des tests
7. Est-ce que le nombre d'occurrence du test est égal à 10 ? Si oui aller à la ligne 13, sinon poursuivre
8. Vérification de la synchronisation des horloges de Alpha et Bravo (voir 5.2.4.2 page 50)
9. Réaliser une occurrence du test
10. Attendre 10 secondes
11. Sauvegarder les résultats expérimentaux de cette occurrence de test
12. Aller à la ligne 8
13. Marquer le test comme réalisé
14. Aller à la ligne 6
15. La série des douze tests est complètement réalisée pour le noyau sélectionné
16. Est-ce que la série des douze tests à été réalisée pour chacun des noyaux ? Si oui, aller à la ligne 19, sinon poursuivre
17. Éteindre les ordinateurs Alpha et Bravo
18. Aller à la ligne 1
19. Tous les résultats expérimentaux sont disponibles : fin de l'expérimentation
20. Éteindre les ordinateurs Alpha et Bravo

La quantité de données expérimentales générée par les tests exige d'automatiser une partie de la manipulation. En conséquence, nous avons automatisé la réalisation des occurrences de test et la sauvegarde des résultats expérimentaux à l'aide d'un script. Il permet de gérer un nombre quelconque de tests et d'occurrences de test. L'ensemble des paramètres expérimentaux peuvent être modifiés dans l'en-tête du script. Nous l'avons conçu comme un outil interactif qui guide l'expérimentateur dans la manipulation. Le script demande deux paramètres à l'expérimentateur : d'une part, le test à réaliser ; d'autre par, l'emplacement où sauvegarder les

résultats expérimentaux. Sur base de ces paramètres, la réalisation des occurrences d'un test et la génération des données expérimentales sont entièrement automatisées. Le script affiche uniquement des informations sur les opérations en cours d'exécution. Il est illustré en annexe G page 133. Nous fournissons ce script pour assurer la transparence sur notre protocole expérimental et faciliter les éventuelles reproductions de l'expérience. Les figures 5.4 et 5.5 illustrent le script en cours d'exécution.

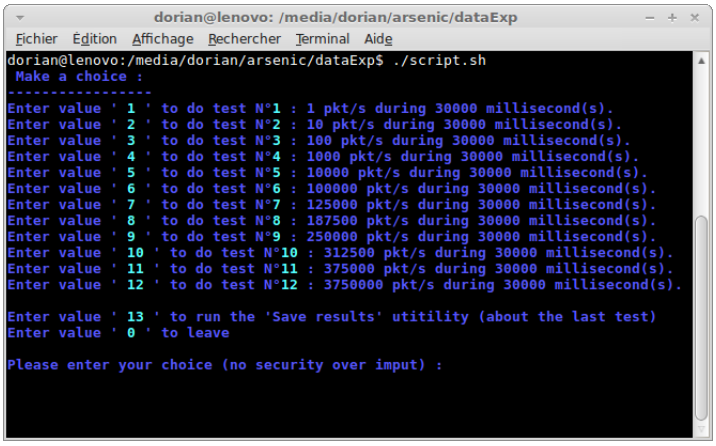

FIGURE 5.4 – Script – Outil interactif d'automatisation des tests

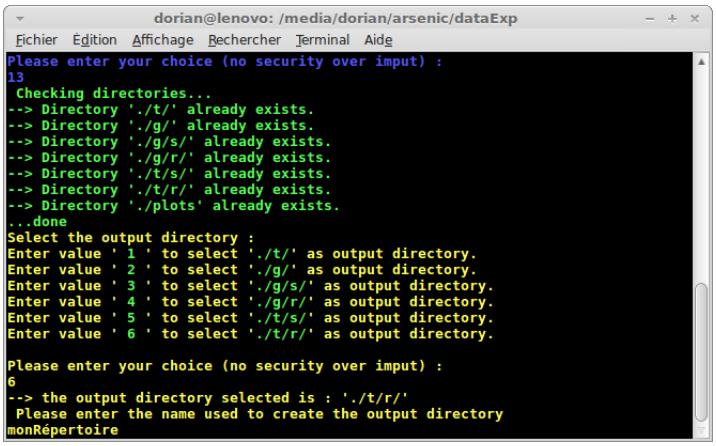

FIGURE 5.5 – Script – Outil interactif de sauvegarde des données expérimentales

5.2.4 Préparation de l'expérience

5.2.4.1 Installation de Debian

Nous avons installé Debian par le logiciel d'installation réseau. Il permet une taille minimale des données nécessaires à l'installation du système d'exploitation. Le logiciel récupère les composants logiciels à installer au fur et à mesure de sa progression au moyen d'une connexion Internet. Les logiciels d'installation traditionnels intègrent l'ensemble des composants logiciels susceptibles d'être installés. Avec Debian, un tel processus d'installation nécessiterait une quantité de données équivalente à quatre DVD de 4,7 GB. Le processus d'installation réseau nécessite

seulement 200 MB de stockage. L'installation du système Debian est une installation par défaut. Nous avons choisi d'équiper le système d'un environnement graphique intitulé "Mate". Nous souhaitions par là obtenir un système minimal et représentatif des équipements informatiques actuels.

5.2.4.2 synchronisation NTP

NTP signifie *"Network Time Protocol"*. C'est un protocole permettant de synchroniser l'horloge des ordinateurs sur un réseau (voir [73, 46, 74]). Nous avons ajouter le logiciel "*NTP*" pour synchroniser les horloges de Alpha et Bravo. C'est une étape nécessaire pour pouvoir calculer les délais de transmission. Si nous n'utilisons pas la synchronisation d'horloge, nous aurions des délais de temps négatifs et donc invalides. Ce fait est reporté dans la documentation de D-ITG (voir [10]). Nous avons définit l'ordinateur Alpha comme serveur NTP. Bravo est un client de ce serveur. Les rôles de Alpha et Bravo sont définis en éditant le fichier de configuration *"/etc/ntp.conf"*. Les commentaires présent dans ce fichier permettent de définit facilement les paramètres.

5.2.4.3 Installation de DITG

DITG est un logiciel livré sous la forme d'une archive. Elle contient le code source du logiciel, des scripts de déploiement et un manuel d'utilisation. Il est nécessaire de compiler et d'installer DITG avant de pouvoir l'utiliser. Le manuel indique la nécessité du compilateur GNU C pour la compilation. Nous avons été amené a installer le compilateur GNU C++ en supplément. Cette dépendance n'est pas renseignée dans le manuel de l'outil. Lorsque ces dépendances sont résolues, la compilation se déroule conformément aux instructions du manuel (voir [10]). L'installation de fait grâce à un script de déploiement. Elle est automatisée.

5.2.4.4 Compilation des noyaux

Nous décrivons dans cette sous-section l'ensemble des opérations mises en oeuvre pour compiler les noyaux généraliste et temps-réel. Dans la suite de nos explications, nous utiliserons le symbole *"∼"* pour représenter le répertoire personnel de l'utilisateur courant. Nous utiliserons Debian 8.1.0 dans son adaptation pour la plateforme AMD64. Il s'agit du jeu d'instructions correspondant à l'architecture de nos ordinateurs. Le noyau de base fourni avec Debian est la version 3.16.0-4. Or, il n'existe pas de version temps réel pour ce noyau. Dès lors nous prenons la dernière version stable du noyau pour laquelle il existe une version temps-réel. En date de l'expérience, il s'agit de la version 4.0.5. La version temps-réel se présente comme un patch à appliquer sur le noyau standard. On parle de patch "RT". Nous utilisons la version "4.0.5-rt4" de ce patch. Nous présenterons la compilation des noyaux étape par étape. Nous commencerons par le noyau généraliste et ensuite le noyau temps-réel. Lorsque des manipulations spécifiques seront nécessaires, nous les représenterons comme ceci :

```
# Nom de la manipulation
Commande 1
Commande i
Commande N
```

Étape 1 :

Nous devons récupérer le code source des noyaux sur le site internet "Kernel.org". Les adresses pour le noyau standard et le patch RT sont disponibles respectivement en [42] et [43]. Nous obtenons deux archives : "linux-4.0.5.tar.xz" et "patches-4.0.5-rt4.tar.xz".

Étape 2 :

D'abord, nous ouvrons un terminal. Ensuite, nous créons un répertoire *"kernel"* à la racine du répertoire personnel de delta. Le fichier "linux-4.0.5.tar.xz" est copié dans le répertoire "kernel". La suite du processus se déroule principalement en ligne de commande au travers de ce terminal. Nous lançons la décompression de l'archive avec la commande :

```
# extraction de l'archive
mkdir ~/kernel
cd ~/kernel
tar -xvf linux-4.0.5.tar.xz
```

Un dossier nommé *"linux-4.0.5"* est alors créé à la racine du dossier *"kernel"*.

Étape 3 :

Certains paquets sont indispensables pour réaliser la compilation du noyau. Ils représentent l'ensemble des applications, logiciels et programmes nécessaires au processus de configuration et de compilation des noyaux. Nous devons posséder les droits de l'utilisateur *"root"* pour réaliser ces opérations. Nous lançons le processus d'installation avec la commande suivante :

```
# Installation des paquets
apt-get install debconf-utils depkg-dev
    debhelper build-essential kernel-package
    libncurses5-dev fakeroot
```

Nous signalons le nombre important de ces paquets. L'outil aptitude (voir *"apt-get"* dans le code) nous indique 92 paquets à installer, pour un total de 753 Mo d'archive à télécharger. Ce qui représente un besoin total de 1.226 Mo une fois installé. Lors de la manipulation, l'outil de configuration des paquets affiche le message suivant : "Une nouvelle version du fichier de configuration du noyau est disponible, la version actuellement utilisée a été modifiée localement. Laquelle souhaitez-vous utiliser ?". Ce message signifie que le fichier de configuration du noyau actuellement utilisé a été personnalisé pour notre architecture. Il est différent du fichier de configuration par défaut. Nous choisissons de garder la version actuellement utilisée.

Étape 4 :

Une fois l'installation des paquets terminée, nous sauvegardons notre fichier de configuration actuel. Le nom peut différer d'une architecture à l'autre. Nous lançons la commande suivante :

```
# Sauvegarde du fichier de configuration
cp /boot/config-3.16.0-4-amd64 ~/config-3.16.0-4-amd64.save
```

Nous obtenons alors une copie du fichier de configuration à la racine de notre répertoire personnel.

Étape 5 :

Nous réalisons une copie de ce fichier de configuration dans le répertoire *"linux-4.0.5"* et nous le renommons *".config"* :

```
# Copie du fichier de configuration
cp ~/config-3.16.0-4-amd64 ./linux-4.0.5/.config
```

Étape 6 :

Nous allons nous placer dans le répertoire " /kernel/linux-4.0.5/ " et lancer l'assistant de configuration du noyau :

```
# Lancement de l'assistant de configuration noyau
cd ~/kernel/linux-4.0.5/
make menuconfig
```

Une interface en ligne de commande s'affiche alors. Nous réutilisons notre fichier de configuration précédent sans modification supplémentaire, nous pouvons enregistrer et quitter l'outil de configuration. Nous retournons à notre ligne de commande. Elle nous indique le message suivant :

```
# Message de fin de configuration
Execute 'make' to start the build or try 'make help'.
```

Étape 7 :

Nous exécutons la compilation du nouveau noyau avec les commandes suivante. Elles doivent être entrées l'une après l'autre. La commande *"Make clean"* permet de supprimer tout résidu d'une compilation précédente et qui pourrait compromettre notre tentative actuelle. La commande *"Make"* va compiler le noyau. C'est une opération très longue. Nous avons constaté une durée de 2h24 pour son exécution complète. Une fois compilé, la commande *"Make modules_install"* va installer les nouveaux modules qui ont été compilés en même temps que le noyau. Pour un rappel sur les modules, consulter la sous-section 3.1 page 31. La commande *"Make install"* intègre le nouveau noyau dans le système d'exploitation :

```
# Commande pour la compilation
Make clean
Make
Make modules_install
Make install
```

Étape 8 :

Le noyau est maintenant installé. Il nous reste une dernière étape. Comme nous l'avons dit à la sous-section 3.3 page 36 que le système d'exploitation Débian offrait le choix du noyau a utiliser au du démarrage du système. Nous allons rendre cette option disponible pour notre nouveau noyau. Il faut exécuter la commande de mise à jour du programme d'amorçage. C'est nécessaire pour sélectionner notre nouveau noyau au démarrage de l'ordinateur. Le programme d'amorçage est le premier chargé en mémoire lors du démarrage d'un ordinateur. Il est assure le placement du système d'exploitation en mémoire. Dans notre cas, ce programme est "GNU Grub 2". Nous exécutons la commande suivante :

```
# Mise a jour du programme de demarrage
update-grub
```

Étape 9 :

Les opérations de compilation du noyau temps réel sont identiques à celles du noyau générique. Cependant, il existe une étape supplémentaire. Elle se trouve entre les étapes 3 et 4 du processus de compilation du noyau générique. Nous devons appliquer le patch sur les sources du noyau générique. Nous Installation le programme 'patch' avec la commande suivante :

```
# Installation du programme patch
apt-get install patch
```

Nous décompressons l'archive *"patches-4.0.5-rt4.tar.xz"* dans le répertoire *"kernel"* :

```
# extraction de l'archive
cd ./kernel
tar -xvf patches-4.0.5-rt4.tar.xz
```

Nous avons alors deux dossiers : l'un avec les sources du noyau ; l'autre contenant le patch à appliquer au noyau. Toujours en ligne de commande, nous nous dirigeons dans le répertoire des sources noyau. Ensuite, nous lançons la commande pour modifier le code source du noyau générique :

```
# modification des sources du noyau generaliste
Cd ~/linux-4.0.5
Patch ../patches/
```

5.2.4.5 L'outil Mogli

Nous avons expliqué dans la sous-section 5.2.3 page 47 que chaque test est réalisé dix fois. Chacune de ces dix occurrences contient une mesure du débit chaque millième de seconde sur une durée de trente secondes. Cela correspond à 30 000 mesures par occurrence de test et donc 300 000 par test. Lors de nos recherche, nous n'avons pas trouvé d'outil permettant de calculer des moyennes arithmétiques et mobile directement à partir des résultats expérimentaux générés par D-ITG. La quantité de données à traiter met en échec les tableurs bureautiques classiques. Microsoft Excel et Open Office Calc ne nous on pas permis d'interpréter ces données. En conséquence, nous avons développé notre propre outil : le logiciel *"Mogli"* qui est l'acronyme de *"moyenne glissante"*. C'est une autre appellation pour la moyenne mobile. Il se présente comme un outil à manipuler en ligne de commande. Nous l'avons rédigé en Java. La figure 5.6 présente une copie d'écran de notre logiciel. Nous l'avons rédiger pour faciliter son utilisation et sa diffusion. Le logiciel est paramétrable et contient les explications nécessaires à son utilisation. La figure 5.6 présente le message d'aide de Mogli. Des informations sur la manipulation du logiciel et sa syntaxe sont visibles sur l'illustration.

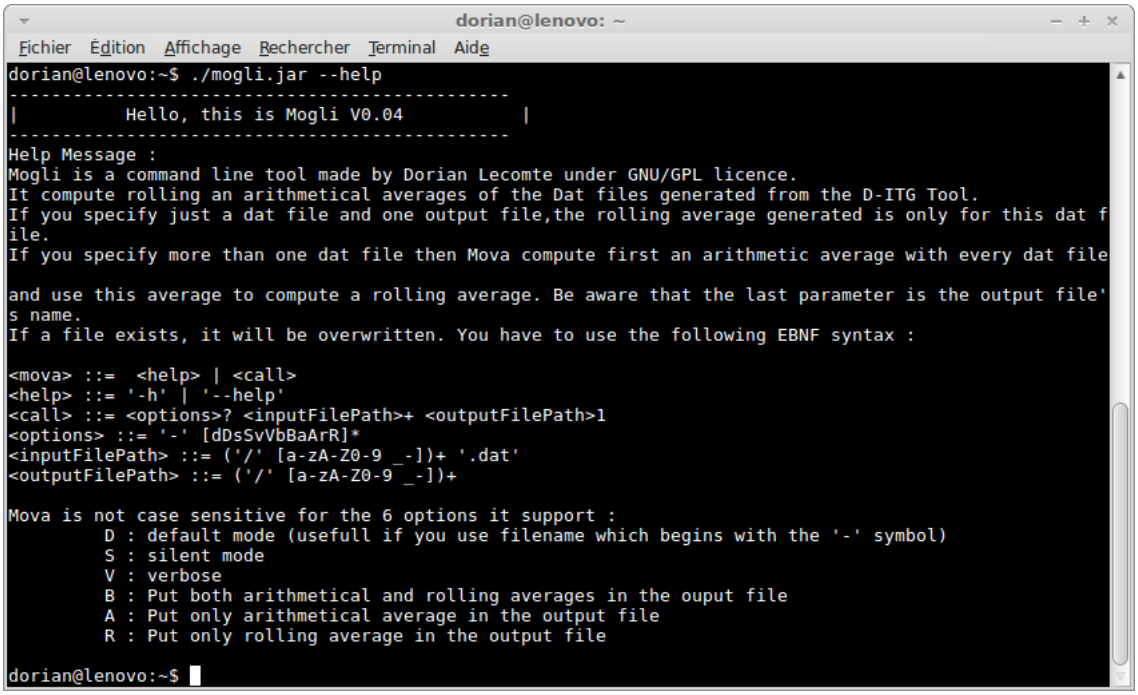

FIGURE 5.6 – Logiciel Mogli – Capture d'écran

5.2.4.6 Déploiement

L'ensemble formé par le système d'exploitation, les noyaux compilés et DITG installé définit notre environnement de travail. Nous avons réalisé une sauvegarde de ce système à l'aide du logiciel Clonezilla (voir [106]). Ce logiciel permet la copie de données d'un disque dur entier ou d'une partition. Nous avons déployé cette sauvegarde sur l'ensemble de nos ordinateurs. Ils sont tous matériellement identiques. C'est ce qui rend le déploiement de la sauvegarde possible. Associer un même ordinateur avec une sauvegarde identique sur chacun d'eux permet d'avoir des clones. Ainsi, nous avons l'assurance d'avoir des ordinateurs rigoureusement identiques en tout point.

5.3 Données et observations

L'ensemble des données expérimentales collectées représente un quantité de données brutes égale à 82,5 Go. Ce sont des fichiers de texte contenant l'ensemble des mesures effectuées pour l'ensemble des métriques développés dans la section 5.2 page 43. Cette quantité de données est trop conséquente pour être représentée dans ce document. Ces données brutes ont étés utilisées pour générer des graphiques. Ils sont consultables à la section 5.5 page 56.

5.4 Calculs

Cette section expose les concepts mathématiques utilisés dans notre analyse. Nous expliquerons d'abord les concepts de moyenne arithmétique et de moyenne mobile. Ensuite, nous montrerons les liens entre les données collectées et ces deux types de moyenne.

La moyenne arithmétique est une valeur statistique qui caractérise un ensemble d'éléments. Elle exprime la valeur que tous les éléments de l'ensemble devraient avoir s'ils étaient tous identiques pour ne pas changer la somme de leurs valeurs respectives. Un exemple type est la moyenne scolaire sans utilisation de la pondération. Elle est notée $\bar{x}$. Pour un ensemble fini de n valeurs discrètes elle se définit :

$$\bar{x} = \frac{x_1+x_2+...+x_n}{n} = \frac{1}{n}\sum_{i=1}^{n} x_i \text{ où } n \in N$$

La moyenne mobile est une autre valeur statistique. Elle se définit sur une suite ordonnée de valeurs. Elle permet de "lisser" les valeurs de la suite en limitant l'impact des écarts entre ces valeurs. C'est une mesure particulièrement utile lorsque les valeurs fluctuent fortement d'une mesure à l'autre. La moyenne mobile définit une moyenne pour des sous-ensembles de termes d'une suite. Il existe différents types de moyennes mobiles. Nous utiliserons la moyenne mobile arithmétique. C'est la plus simple et la plus répandue. Soit une suite finie (S_n) *avec* $n \in N$. La moyenne mobile abrégée MM_j pour l'élément S_j de la suite S_n se définit comme la moyenne arithmétique des J premiers éléments de cette liste avec $j \in N \wedge (1 \leq j \leq n)$. Ainsi, nous obtenons :

$$MM_j = \frac{1}{j}\sum_{i=1}^{j} S_i \text{ où } i \in N \wedge (j \leq n \in N)$$

Nous allons maintenant faire le lien entre le calculs des moyennes et nos données expérimentales. Chaque test contient dix occurrences de test qui sont chacune représentée par une suite réelle finie nommée O et indicée d'un nombre i permettant de la différencier des autres occurrences :

$$\forall\ test\ \exists!\ \{(O_{i_n})|(i, n \in N) \wedge (1 \leq i \leq 10) \wedge n = 30000\}$$

Sur base des occurrences d'un test, nous pouvons calculer la moyenne arithmétique d'un test. Nous effectuons le calcul de la moyenne arithmétique pour chaque mesure de débit. L'ensemble des moyennes calculées sont définies dans une suite réelle finie notée (MA_n) avec $n \in N$. Nous obtenons :

$$MA_k = \frac{1}{10}\sum_{i=1}^{10} O_{i_k} \text{ où } (i, k \in N) \wedge (1 \leq k \leq n)$$

Nous pouvons maintenant calculer la moyenne mobile. Elle est calculée pour chaque valeur de débit. L'ensemble des moyennes calculées sont définies dans une suite réelle finie notée (MM_n) avec $n \in N$. Nous obtenons :

$$MM_k = \frac{1}{k}\sum_{i=1}^{k} MA_i \text{ où } (i, k \in N) \wedge (1 \leq k \leq n)$$

La moyenne mobile nous permettra de faire des observations entre le débit spécifié des tests et le débit du trafic mesuré. Nous serons capables de déterminer si le débit du trafic généré par D-ITG est statistiquement conforme aux spécifications des douze tests.

5.5 Résultats

Nous présenterons nos résultats en quatre sous-sections. La sous-section 5.5.1 présentera d'abord notre analyse des débits. Ensuite, la sous-section 5.5.2 page 65 exposera l'analyse des délais. Après, la sous-section 5.5.4 page 68 expliquera l'analyse des datagrammes. Enfin, la sous-section 5.5.3 page 67 présentera une analyse de la gigue. L'ensemble des graphiques exposés sont de taille réduite pour nous permettre la mise en place de commentaires. L'annexe F page 115 présente chacun des graphiques sur une page entière. Nous donnerons les liens vers les pages annexes concernées au fur et à mesure de nos explications.

5.5.1 Analyse des débits

Nous commençons notre analyse des débits par une vue d'ensemble des résultats expérimentaux. La figure 5.7 illustre l'ensemble des débits moyens mesurés sur Alpha et Bravo pour toutes les occurrences des tests [1]. L'axe des ordonnées exprime le débit en Kb/s. Sa graduation maximale correspond à la bande passante maximale du réseau. L'axe des abscisses indique les numéros de chaque test. Le graphique contient deux types d'informations. D'une part toute mesure du débit moyen est représentée comme un point. Pour un rappel sur la mesure du débit moyen, consulter la sous-section 5.2.3 page 47. D'autre part, la moyenne arithmétique des débits moyens est représentée sous la forme d'une ligne. Ces deux informations sont disponibles pour le noyau généraliste et le noyau temps réel. Pour ces deux noyaux, les débits mesurés par Alpha et Bravo se recouvrent parfaitement. Les résultats expérimentaux générés par D-ITG coté émetteur et récepteur sont cohérents.

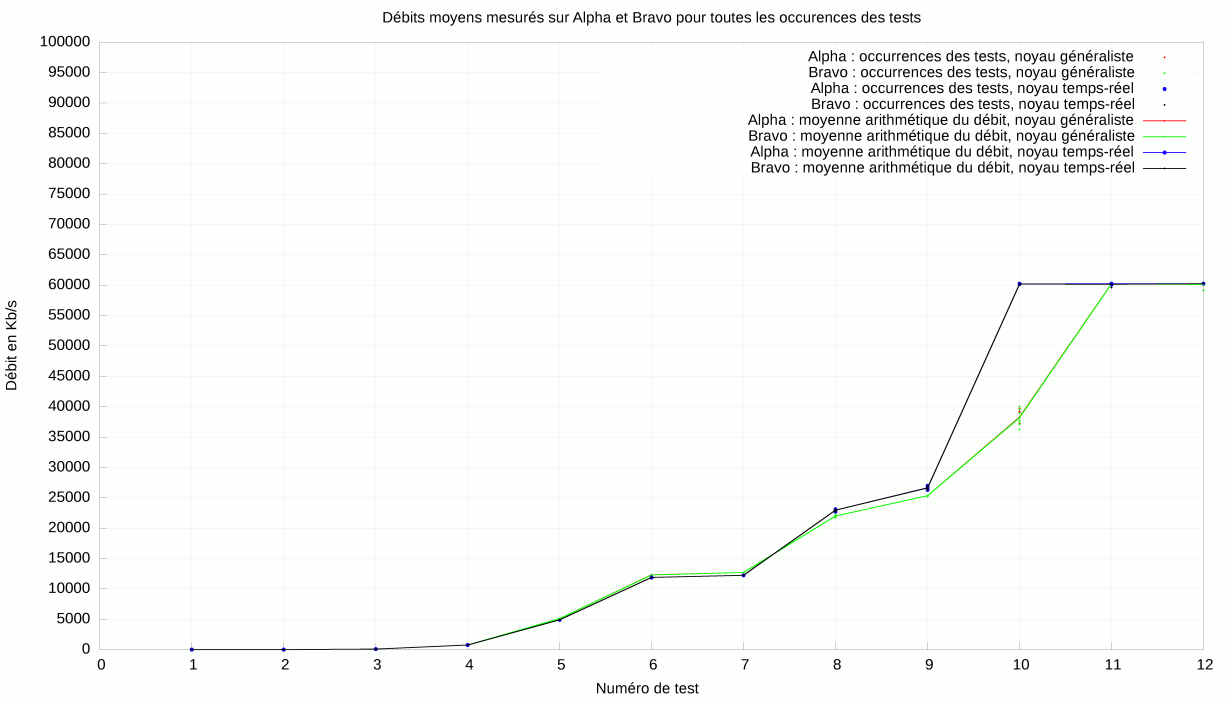

FIGURE 5.7 – Analyse des débits – Vue d'ensemble du débit

1. Graphique taille réelle disponible en annexe F page 127

Le graphique montre que les débits moyens entre le noyau généraliste et le noyau temps-réel sont identiques jusqu'au test 3 inclus. Ensuite, le noyau générique montre un débit moyen supérieur à celui du noyau temps-réel sur les tests 4 et 7. Après, le graphique illustre un point de croisement entre les tests 7 et 8. Dès cet instant, le débit moyen du noyau temps-réel est supérieur à celui du noyau générique jusqu'au test 10 inclus. Les tests 10 et 11 montrent un débit moyen identique pour les deux noyaux. Ce débit correspond aux limites des performances du matériel utilisé.

Nous allons maintenant analyser individuellement les résultats de chaque test. Les résultats ont étés synthétisés dans une série de douze graphiques reprenant les observations faites sur le noyau généraliste et le noyau temps-réel. Nous avons représentés deux types d'informations pour chaque noyau. D'une part la moyenne arithmétique des débits instantanés des occurrences du test. D'autre part, la moyenne mobile du débit calculée à partir de la moyenne arithmétique des débits instantanés. Pour un rappel sur la méthode de calcul utilisée, consulter la sous-section 5.4 page 54. Pour chaque graphique, nous adopterons l'échelle de représentation la plus adéquate. Nous souhaitons faciliter la lecture des résultats. En particulier l'observation des différences entre le débit spécifié et le débit mesuré. Sur les graphiques, la moyenne mobile du noyau généraliste est représentée par une courbe sur laquelle se trouve des ronds de couleur bleu. Celle du noyau temps-réel est noire sans motif. Cette représentation permet de distinguer les courbes lorsqu'elles se superposent.

Les trois premiers tests illustrent des performances identiques pour les deux noyaux. Les moyennes mobiles sont identiques pour le noyau généraliste et le noyau temps-réel. Le débit du trafic mesuré correspond au débit spécifié. Les deux noyaux affichent des performances identiques et fidèles aux spécifications jusqu'à un débit de 80 Kb/s. Les graphiques correspondant aux tests 1, 2 et 3 sont illustrés respectivement aux figures 5.8, 5.9 et 5.10 [2].

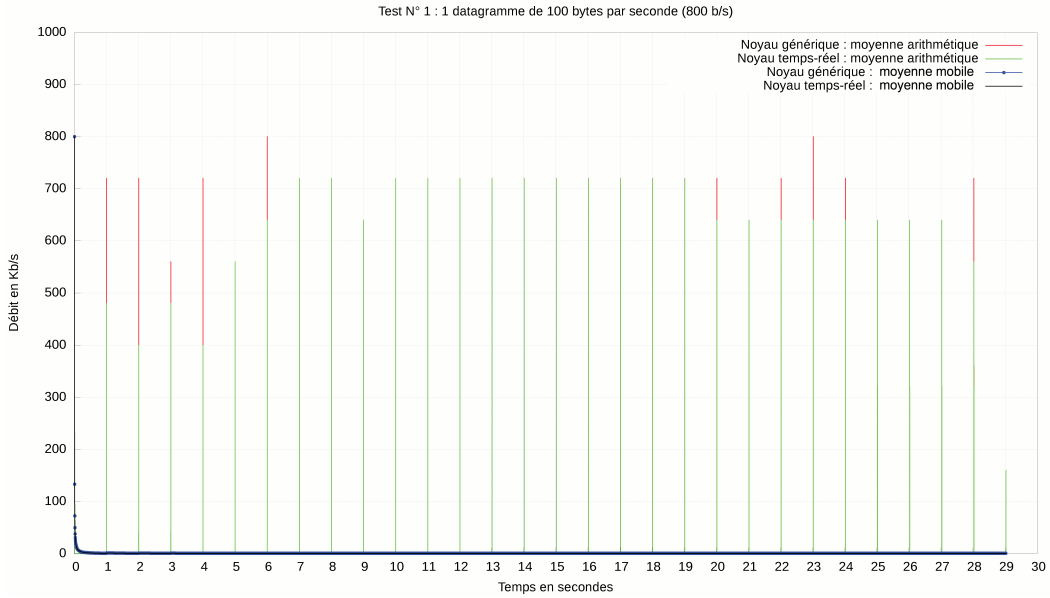

FIGURE 5.8 – Analyse des débits – Test 1

2. Graphiques taille réelle disponibles en annexe F pages 115, 116 et 117

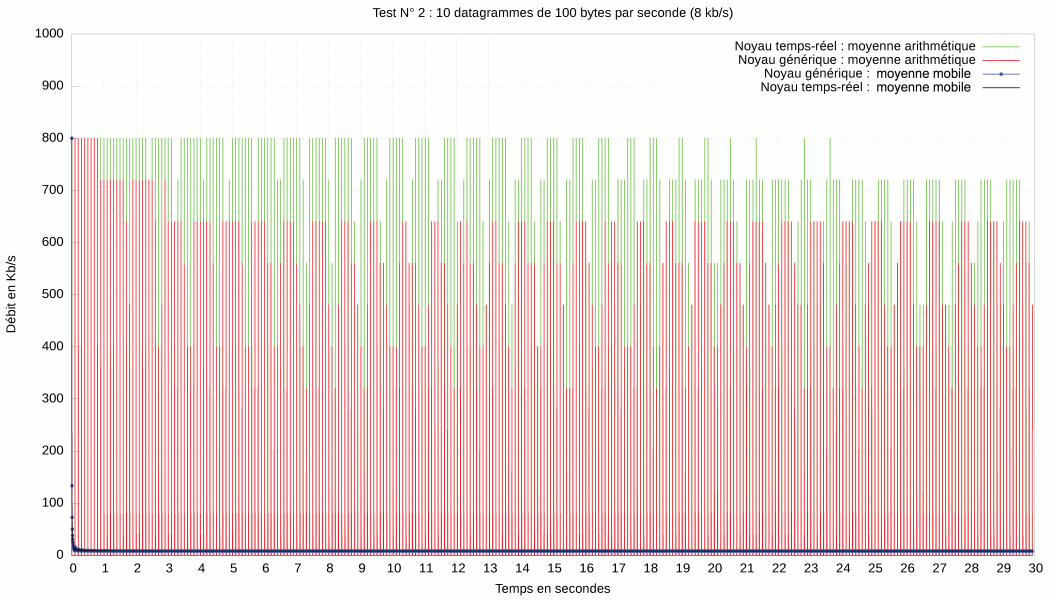

FIGURE 5.9 – Analyse des débits – Test 2

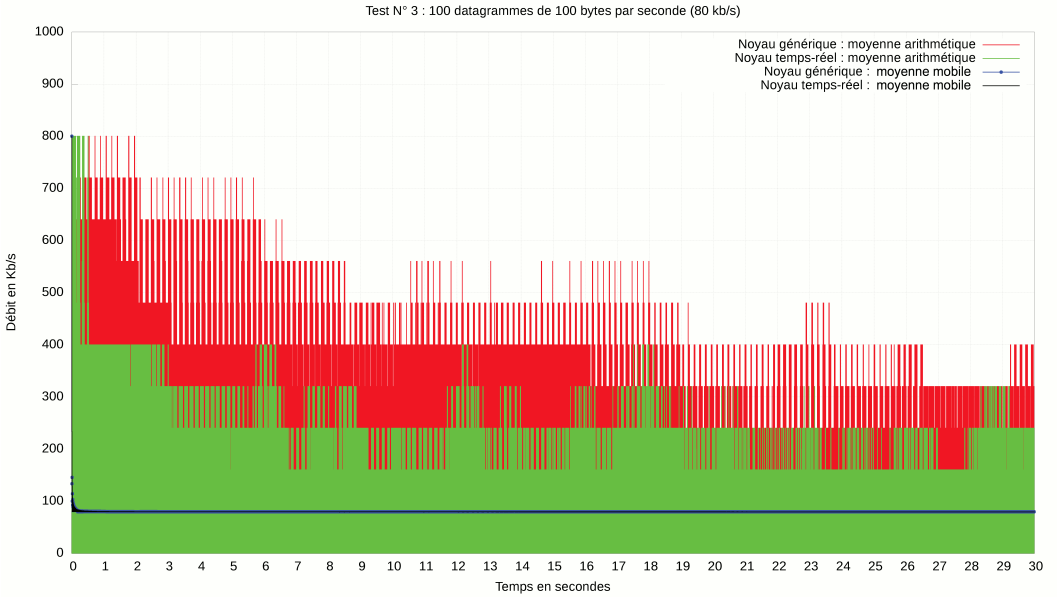

FIGURE 5.10 – Analyse des débits – Test 3

Le test 4 illustre une différence entre le débit du trafic mesuré et le débit spécifié. La figure 5.11 présente le graphique du test 4[3]. En effet, pour un débit spécifié de 800 Kb/s, nous obtenons un débit proche de 755 Kb/s. De plus, les deux noyaux montrent des performances différentes. La moyenne mobile du noyau généraliste se stabilise à une valeur de 760 Kb/s et celle du noyau temps-réel à 750 kb/s. L'écart entre le débit des noyaux est de 10 Kb/s. Le noyau généraliste permet de produire un débit égal à 95% du débit spécifié et le noyau temps réel un débit égal à 93,4 %. La moyenne arithmétique du noyau temps-réel possède une limite

3. Graphique taille réelle disponible en annexe F page 118

supérieur à 800 kb/s. Le test 4 marque la fin de l'adéquation entre le débit du trafic mesuré et le débit spécifié. Pour tous les tests suivants, le débit du trafic mesuré sera systématiquement inférieur au trafic spécifié.

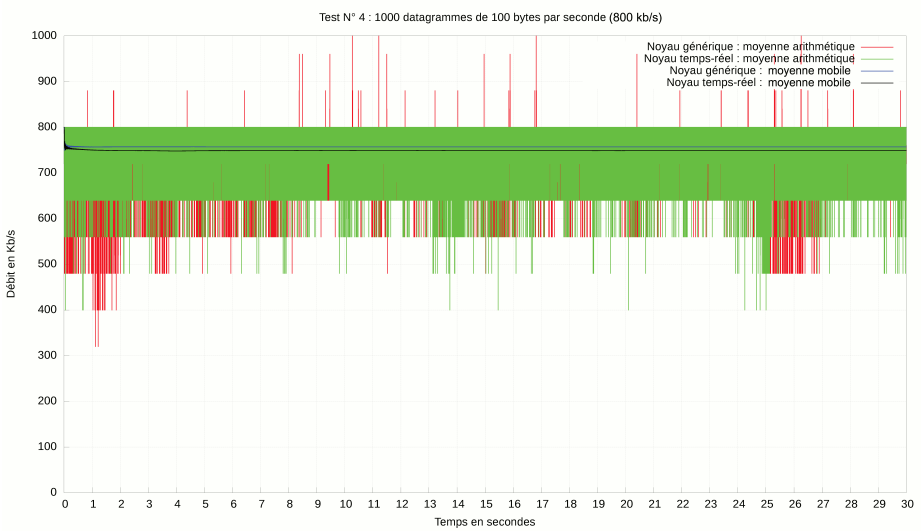

FIGURE 5.11 – Analyse des débits – Test 4

Le test 5 illustre une différence plus importante entre le débit du trafic mesuré et le débit spécifié que le test 4. Pour un débit spécifié de 8 Mb/s, nous obtenons un débit de 5 Mb/s pour le noyau généraliste et un débit de 4,8 Mb/s pour le noyau temps-réel. L'écart entre le débit des noyaux est de 200 Kb/s. Le noyau généraliste permet de produire un débit égal à 62,5% du débit spécifié et le noyau temps réel un débit égal à 60 %. Pour ce test, le noyau généraliste est plus performant que le noyau temps réel. La stabilisation de la moyenne mobile du noyau temps-réel est presque instantanée. Celle du noyau généraliste se stabilise après 3 secondes. La figure 5.12 présente le graphique du test 5 [4].

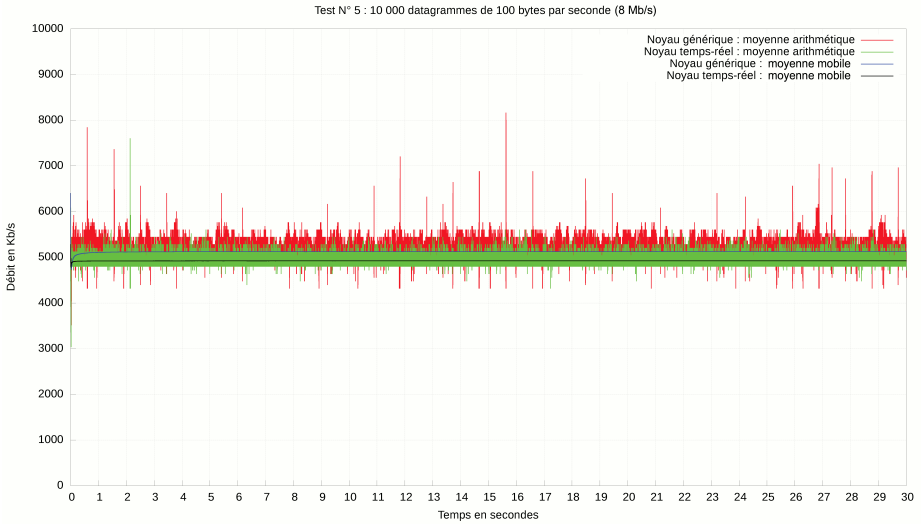

FIGURE 5.12 – Analyse des débits – Test 5

4. Graphique taille réelle disponible en annexe F page 119

Le test 6 illustre une accentuation de la différence entre le débit du trafic mesuré et le débit spécifié. Nous avons déjà constaté ce phénomène aux figure 5.11 et 5.12 qui correspondent respectivement aux test 4 et 5. Pour un débit spécifié de 80 Mb/s, nous obtenons un débit de 12,33 Mb/s pour le noyau généraliste et un débit de 12 Mb/s pour le noyau temps-réel. L'écart entre le débit des noyaux est de 333 Kb/s. Le noyau généraliste permet de produire un débit égal à 15,41 % du débit spécifié et le noyau temps réel un débit égal à 15 %. Le noyau généraliste est ici aussi plus performant que le noyau temps réel. La stabilisation de la moyenne mobile du noyau temps-réel est presque instantanée. Celle du noyau généraliste se stabilise à nouveau après 3 secondes. La figure 5.13 présente le graphique du test 6 [5].

FIGURE 5.13 – Analyse des débits – Test 6

Le test 7 illustre ici encore une accentuation de la différence entre le débit du trafic mesuré et le débit spécifié. Ce phénomène est également remarqué sur les figures 5.11 et 5.12 et 5.13 qui correspondent respectivement aux test 4 à 6. Pour un débit spécifié de 100 Mb/s, nous obtenons un débit de 12,66 Mb/s pour le noyau généraliste et un débit de 12,22 Mb/s pour le noyau temps-réel. L'écart entre le débit des noyaux est de 444 Kb/s. Le noyau généraliste permet de produire un débit égal à 12,66 % du débit spécifié et le noyau temps réel un débit égal à 12,22 %. Le noyau généraliste est ici aussi plus performant que le noyau temps réel. La stabilisation de la moyenne mobile du noyau temps-réel est presque instantanée. Celle du noyau généraliste se stabilise ici encore après 3 secondes. La figure 5.14 page 61 présente le graphique du test 7 [6].

Pour le test 8 et les suivants, le débit spécifié est supérieur à la bande passante du réseau. Or, le réseau n'est jamais saturé à 100 % de ses capacités. Le graphique de la figure 5.7 page 56, illustrait une saturation aux environs de 60 Mb/s. Ce débit n'est atteint que sur les test 11 et 12. Pour les tests 8 à 10, le débit du trafic mesuré augmente progressivement sans jamais

5. Graphique taille réelle disponible en annexe F page 120
6. Graphique taille réelle disponible en annexe F page 121

atteindre le débit spécifié. Nous observons une inversion des performances entre les noyaux. En effet, pour un débit spécifié de 150 Mb/s, nous obtenons un débit de 22 Mb/s pour le noyau généraliste et 23 Mb/s pour le noyau temps-réel. L'écart entre le débit des noyaux est de 1 Mb/s. Le noyau généraliste permet de produire un débit égal à 14,67 % du débit spécifié et le noyau temps réel un débit égal à 15,33 %. Le noyau temps-réel est plus performant que le noyau généraliste. La stabilisation de la moyenne mobile du noyau temps-réel survient après 0,5 seconde. Celle du noyau généraliste se stabilise après 2,5 secondes. La figure 5.15 page 61 présente le graphique du test 8 [7].

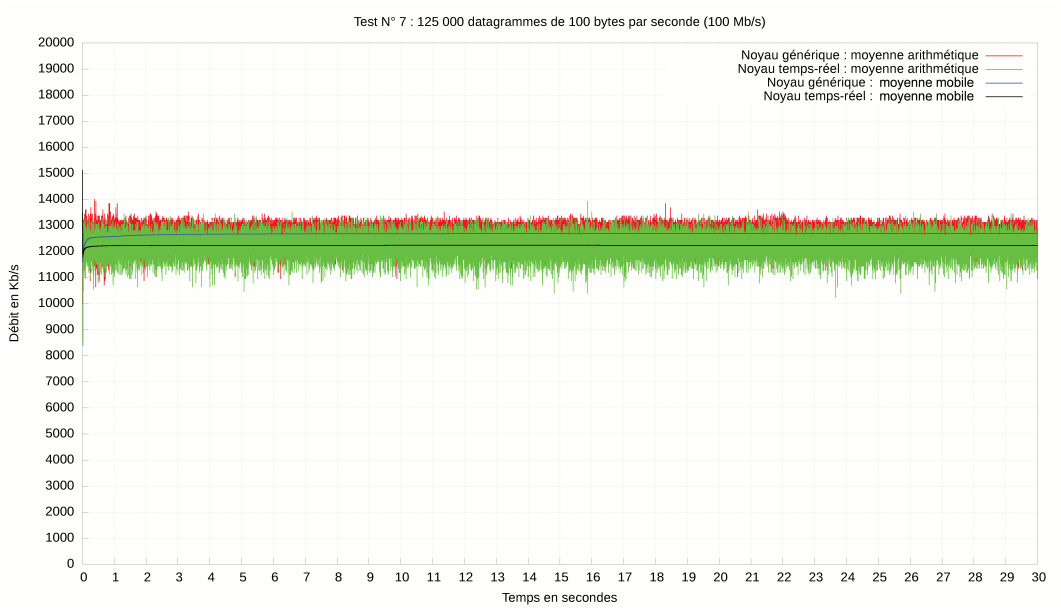

FIGURE 5.14 – Analyse des débits – Test 7

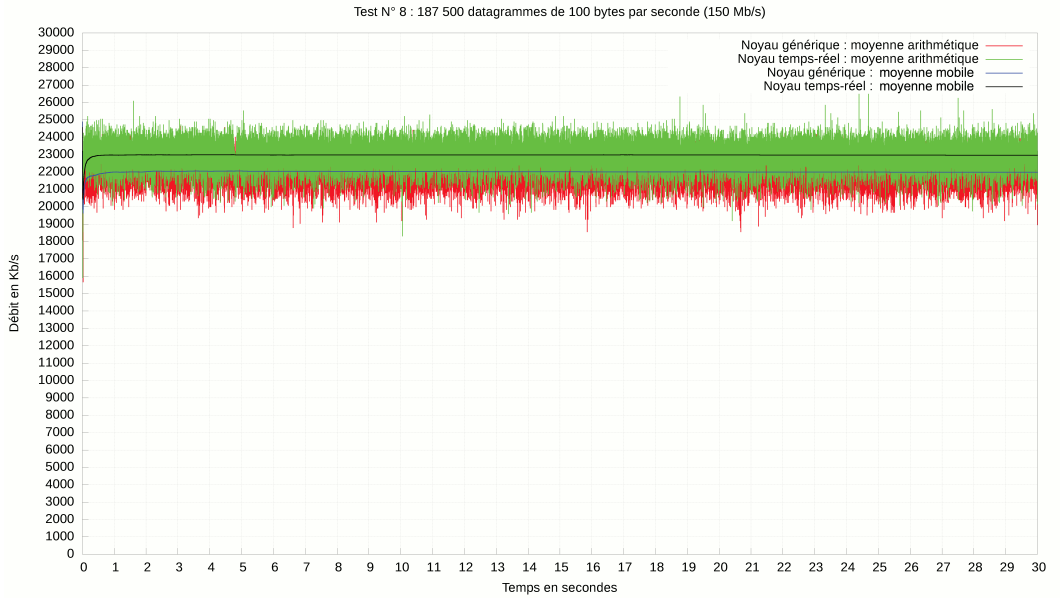

FIGURE 5.15 – Analyse des débits – Test 8

7. Graphique taille réelle disponible en annexe F page 122

Lors du test 9, pour un débit spécifié de 200 Mb/s nous avons obtenu un débit de 25 Mb/s pour le noyau généraliste et un débit de 26 Mb/s pour le noyau temps-réel. L'écart entre le débit des noyaux est de 1 Mb/s. Cette valeur est identique à celle constatée au test 8. Le noyau généraliste permet de produire un débit égal à 12,5 % du débit spécifié et le noyau temps réel un débit égal à 13 %. Le noyau temps-réel est ici aussi plus performant que le noyau généraliste. La stabilisation de la moyenne mobile du noyau temps-réel survient après 0,5 seconde. Celle du noyau généraliste se stabilise après 2 secondes. La figure 5.16 présente le graphique du test 9 [8].

Le test 10 illustre la plus grande différence de performance entre les deux noyaux. le noyau temps-réel génère un débit de l'ordre de 60 % supérieur au noyau généraliste. Pour un débit spécifié de 250 Mb/s nous avons obtenu un débit de 37,5 Mb/s pour le noyau généraliste et un débit de 60 Mb/s pour le noyau temps-réel. Un débit de 60 Mb/s est le débit maximum accessible avec le matériel utilisé. C'est une constatation empirique. L'écart entre le débit des noyaux est de 22,5 Mb/s. Le noyau généraliste permet de produire un débit égal à 15 % du débit spécifié et le noyau temps réel un débit égal à 24%. La stabilisation de la moyenne mobile du noyau temps-réel est presque instantanée. En revanche, celle du noyau généraliste ne se stabilise pas. La figure 5.17 page 63 présente le graphique du test 10 [9].

Les tests 11 et 12 illustrent des résultats identiques. Le test 11 spécifiait un débit de 300 Mb/s et le test 12 un débit de 3 Gb/s. Dans les deux tests, le débit atteint par les deux noyaux est de 60 Mb/s et leurs moyennes mobiles se superposent instantanément. Ces tests montrent que le débit atteint est le débit maximum pour notre montage expérimental. Les figures 5.18 page 63 et 5.19 page 64 illustrent respectivement les graphiques des tests 11 et 12 [10].

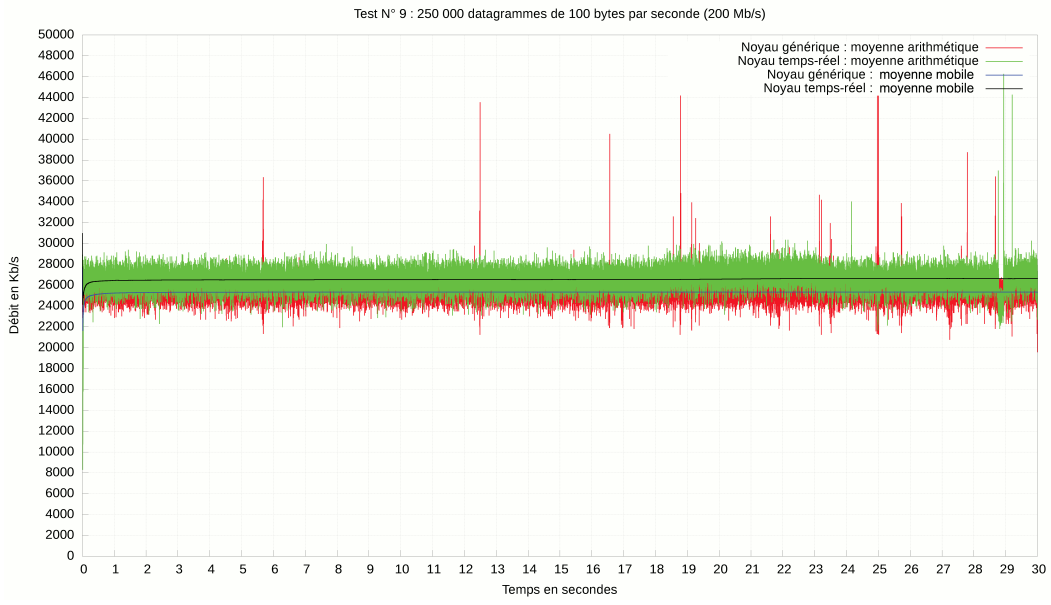

FIGURE 5.16 – Analyse des débits – Test 9

8. Graphique taille réelle disponible en annexe F page 123
9. Graphique taille réelle disponible en annexe F page 124
10. Graphiques taille réelle disponibles en annexe F pages 125 et 126

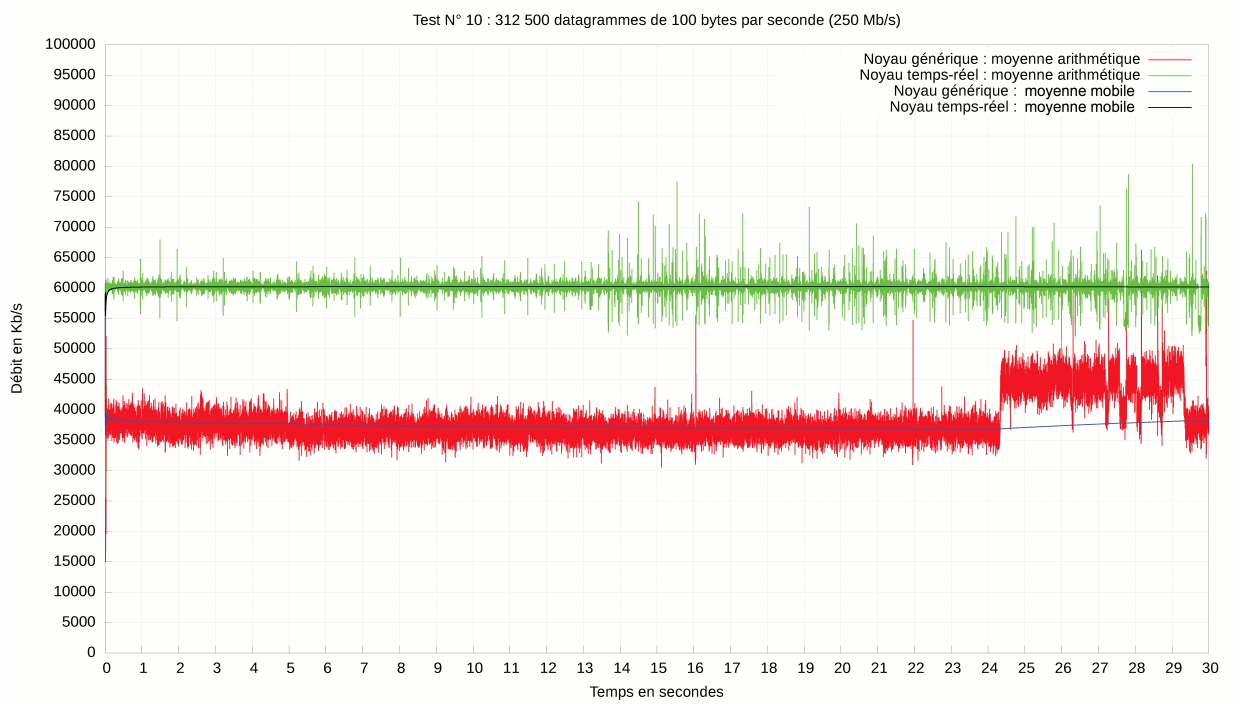

FIGURE 5.17 – Analyse des débits – Test 10

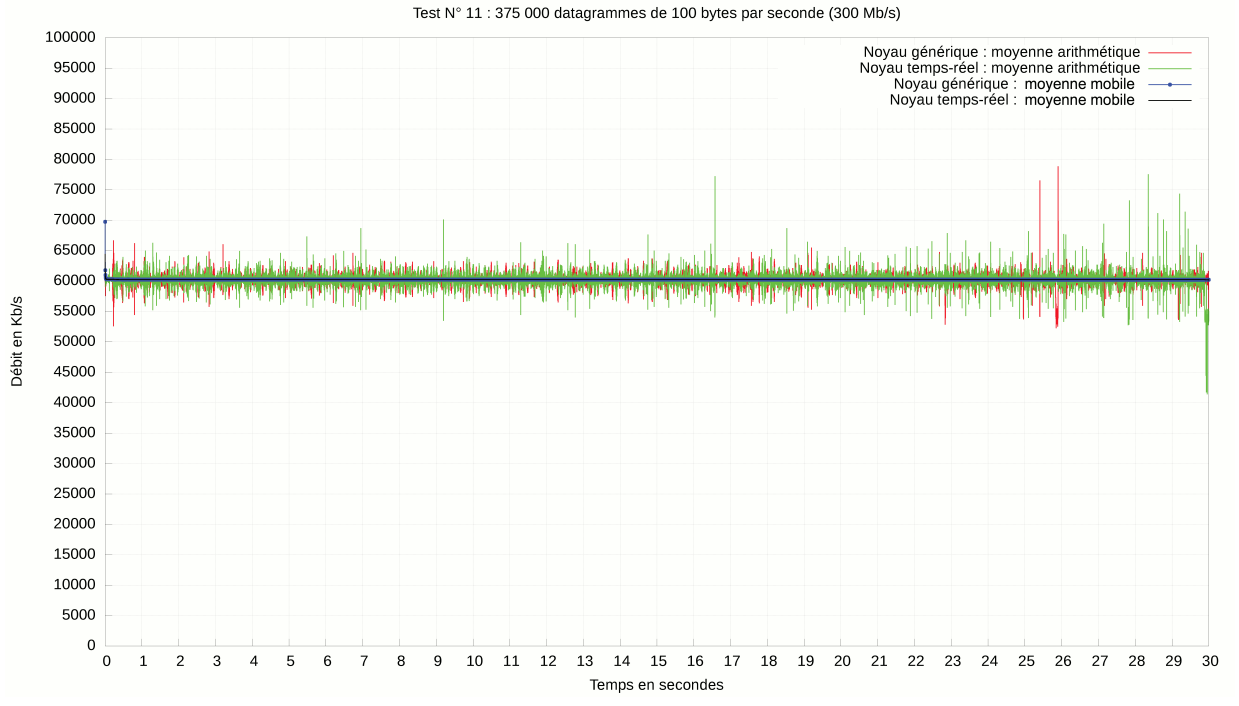

FIGURE 5.18 – Analyse des débits – Test 11

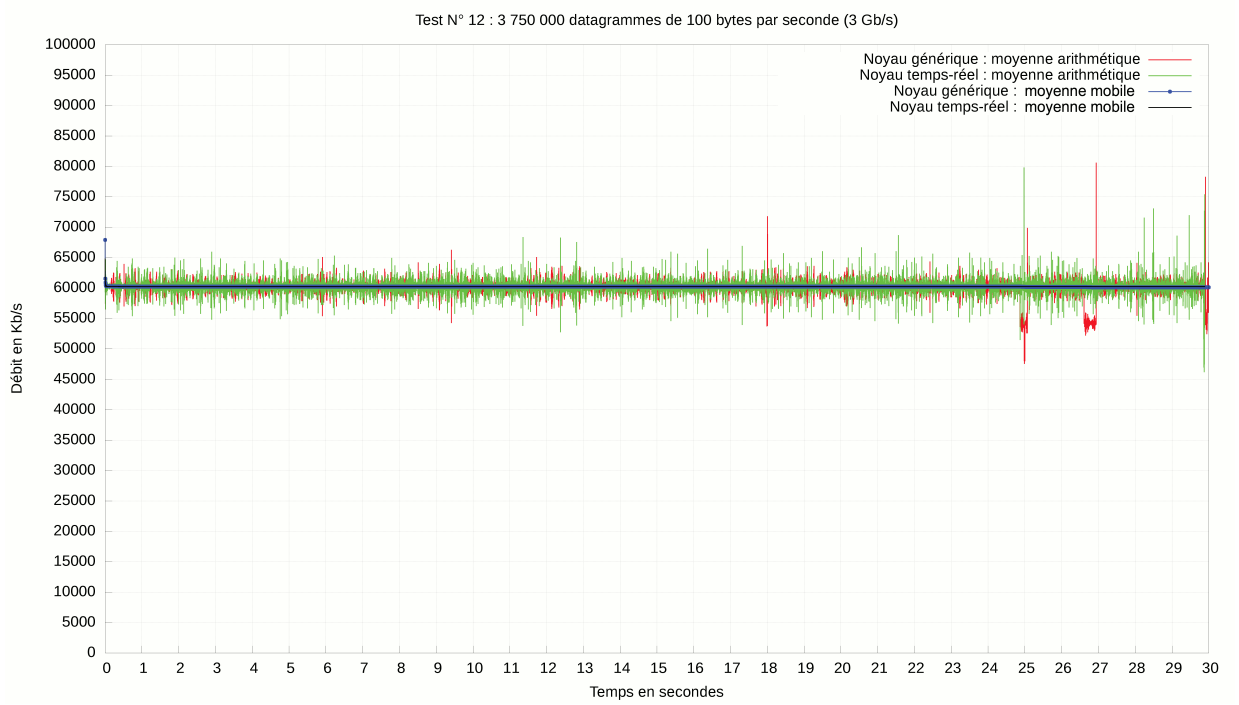

FIGURE 5.19 – Analyse des débits – Test 12

5.5.2 Analyse des délais

L'analyse des délais est constituée de deux graphiques : l'un pour le noyau généraliste ; l'autre pour le noyau temps-réel. Pour chaque graphique, l'axe des ordonnées est gradué en seconde et l'axe des abscisses contient les numéros de test. Les graphiques contiennent deux types d'informations. D'une part, les délais minimums, maximums et moyens calculés pour chaque occurrence des tests. Ces informations sont représentées sous forme de points. D'autre part, la moyenne arithmétique de ces délais calculées pour chaque test à partir des occurrences des tests. Ces moyennes sont représentées sous la forme d'une courbe. Pour chacun des trois types de délais, un code couleur est présent :

— Délais minimums : rouge
— Délais moyens : vert
— Délais maximums : bleu

Le graphique se rapportant aux délais du noyau temps-réel est présenté à la figure 5.20 page 66. L'autre, concernant le noyau généraliste, est illustré à la figure 5.21 page 66 [11]. Les délais obtenus avec le noyau généraliste sont stables jusqu'au test 7. On observe une augmentation de la moyenne des délais maximums au test 6 avec une valeur de 0,2 seconde. Ensuite, les délais sont en augmentation continue sur les tests 7 à 12. Les délais mesurées avec le noyau temps-réel sont plus stables. La moyenne des délais moyens se trouve toujours sous les 0,005 secondes. Jusqu'au test 10, cette moyenne est égal de 0. Le test 6 montre également une augmentation de la moyenne des délais maximums pour une valeur de 0,1 seconde. La moyenne des délais maximums possède une valeur similaire pour les deux noyaux au test 12. Cette valeur est égal de 0,05 seconde. Les moyennes des délais minimums, moyens et maximums sont similaire sur les tests 2, 3 et 4 pour les deux noyaux. Enfin, les différentes mesures du délai maximum sont plus éparpillées pour le noyau généraliste que le noyau temps réel.

11. Graphiques taille réelle disponibles en annexe F pages 128 et 129

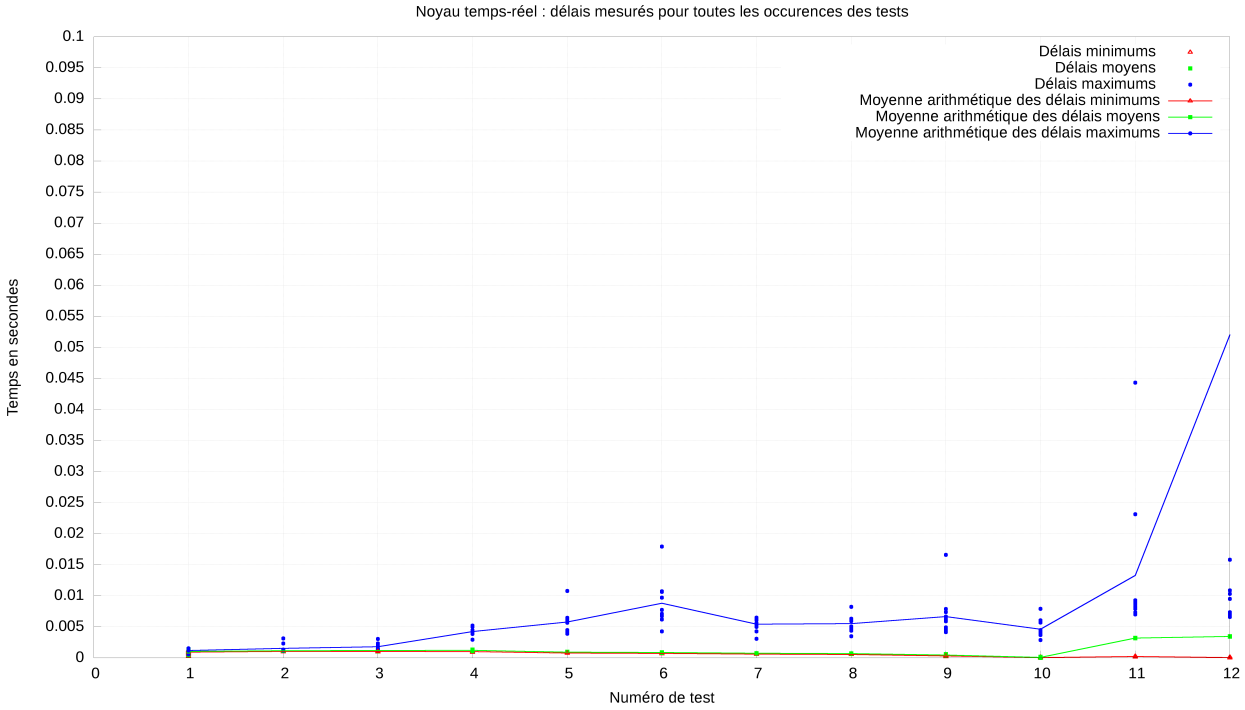

FIGURE 5.20 – Analyse des délais – Noyau temps-réel

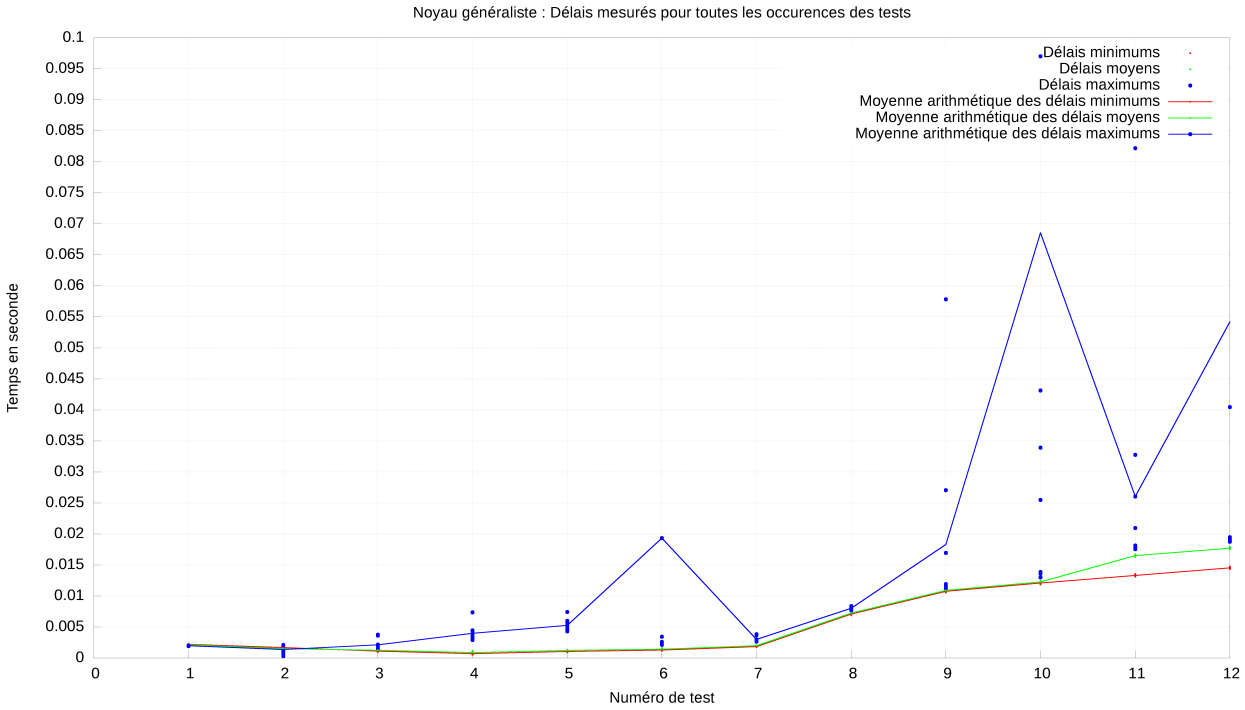

FIGURE 5.21 – Analyse des délais – Noyau généraliste

5.5.3 Analyse de la gigue

L'analyse de la gigue contient un graphique présenté à la figure 5.22 page 67. Pour un rappel sur la définition de la gigue, consulter la sous-section 2.3.1.1 page 18 [12]. Sur ce graphique, l'axe des ordonnées est gradué en seconde et l'axe des abscisses contient les numéros des tests. Il représente deux informations. D'une part toutes les mesures de la gigue effectuées sur chaque occurrence des tests. Elles sont représentées sous forme de triangles rouges pour le noyau généraliste et sous forme de carrés verts pour le noyau temps-réel. D'autre part, la moyennes arithmétique des différentes mesures de la gigue par test. Elle est calculée sur base des différentes valeurs obtenues sur chaque occurrence des tests. Une courbe rouge représente la moyenne arithmétique de la gigue pour le noyau généraliste. Une courbe verte représente celle du noyau temps-réel. Pour observer des différences dans les mesures effectuées avec ces deux noyaux, une résolution d'analyse de l'ordre du cent-millième de seconde est nécessaire. Dans la pratique, ces différences de mesure ne sont pas significatives. L'écart observé au test 10 est à mettre en corrélation avec l'analyse des débits réalisée pour ce test (voir figure 5.17 page 63).

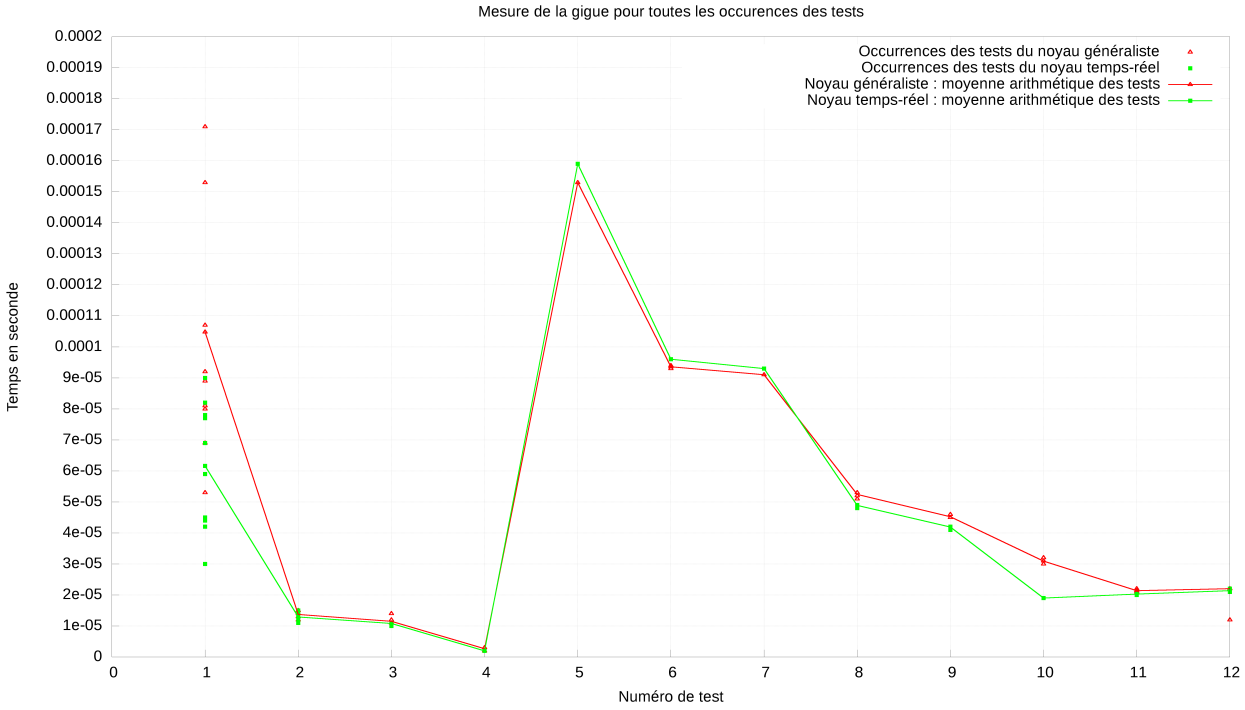

FIGURE 5.22 – Analyse de la gigue – Noyau généraliste et noyau temps-réel

12. Graphique taille réelle disponible en annexe F pages 130

5.5.4 Analyse des datagrammes

L'analyse des datagrammes est composée de deux graphiques. Le premier illustre le nombre de datagrammes transférés pour toutes les occurrences des tests. Il contient deux types d'informations. D'une part, le nombre de datagrammes transférés pour chaque occurrence des tests est représenté sous forme de points. Les mesures relatives au noyau généraliste sont représentées sous forme de triangles rouges ; les données relative au noyau temps-réel sous forme de carrés verts. D'autre part, la moyenne arithmétique du nombre de datagrammes transférés par test. Elle est calculée sur base des différentes valeurs obtenues sur chaque occurrence des tests. Une courbe rouge représente la moyenne du noyau généraliste. Une courbe verte représente celle du noyau temps-réel. Ce graphique est présenté à la figure 5.23. Le second graphique représente le nombre de datagrammes perdus pour toutes les occurrences des tests. Il contient deux types d'informations. D'une part, le nombre de datagrammes perdu pour chaque occurrence des tests est représenté sous forme de points. Les mesures relatives au noyau généraliste sont représentées sous la forme de ronds bleus ; les données relatives au noyau temps-réel sous forme de disques noirs. Ce graphique est disponible à la figure 5.24 page 69.

Le graphique en figure 5.23 possède un aspect très similaire au graphique en figure 5.7 page 56. Le nombre de datagrammes transférés est similaire pour les deux noyau sur les tests 1 à 3. Ensuite, le noyau généraliste montre des performances supérieurs à celles du noyau temps-réel sur les tests 4 à 7. Après, il survient une inversion des tendances entre le test 7 et le test 8. Le noyau temps-réel transfert un nombre de datagrammes plus élevé que le noyau généraliste sur les tests 8 à 10. Ici encore, les tests 11 et 12 montrent des performances identiques sur les deux noyaux. Le nombre de datagrammes transférés y atteint son maximum pour notre montage expérimental.

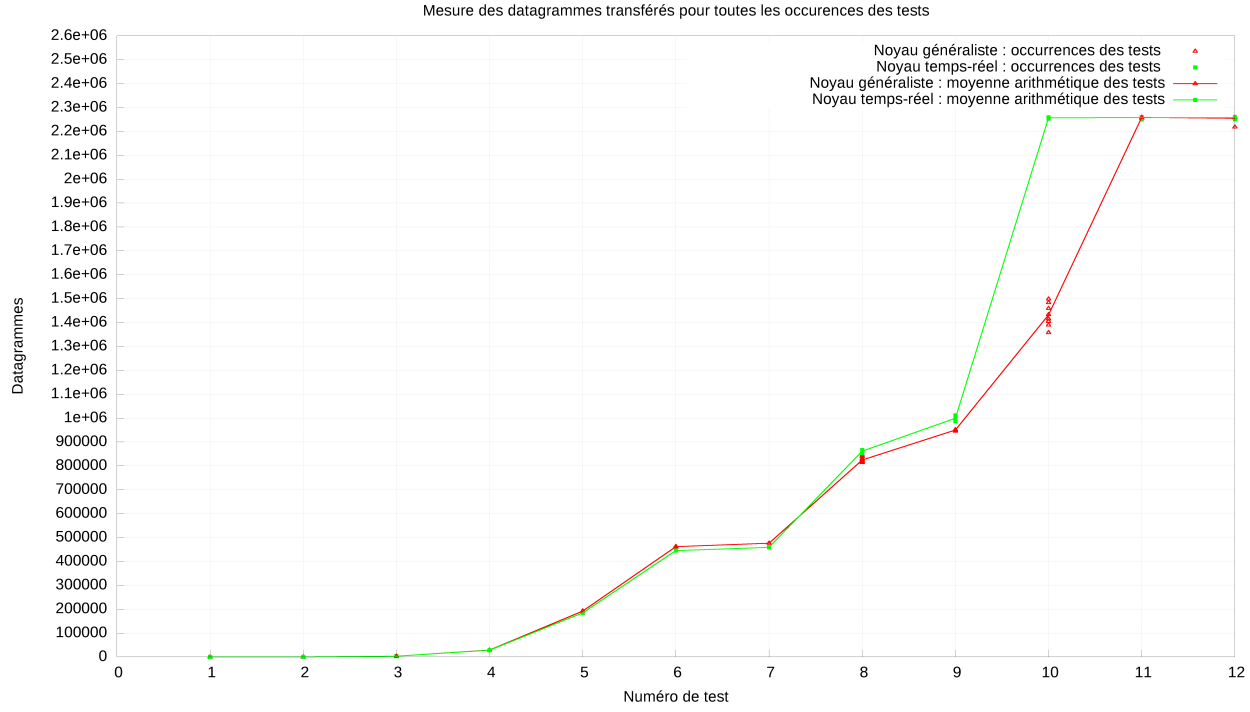

FIGURE 5.23 – Totaux des datagrammes transmis – Noyau généraliste et noyau temps-réel

Le graphique en figure 5.24 montre que le noyau temps-réel ne subit aucune perte de datagramme sur les huit premiers tests. Le noyau généraliste subit des pertes de l'ordre de 250 datagrammes sur le test 6. Pour le test 9, le noyau temps-réel présente une moyenne de 500 datagrammes perdus et le noyau généraliste une perte de 100 datagrammes en moyenne. Le test 10 illustre des moyennes de perte plus importantes pour les deux noyaux : 3250 pour le noyau généraliste et 1600 pour le noyau temps-réel. La moyenne des pertes pour le test 11 est très similaire pour les deux noyaux : 500 datagrammes perdus pour le noyau généraliste et 400 pour le noyau temps-réel. Le test 12 illustre l'écart de plus important : 250 datagrammes perdus pour le noyau temps-réel et 4250 pour le noyau généraliste.

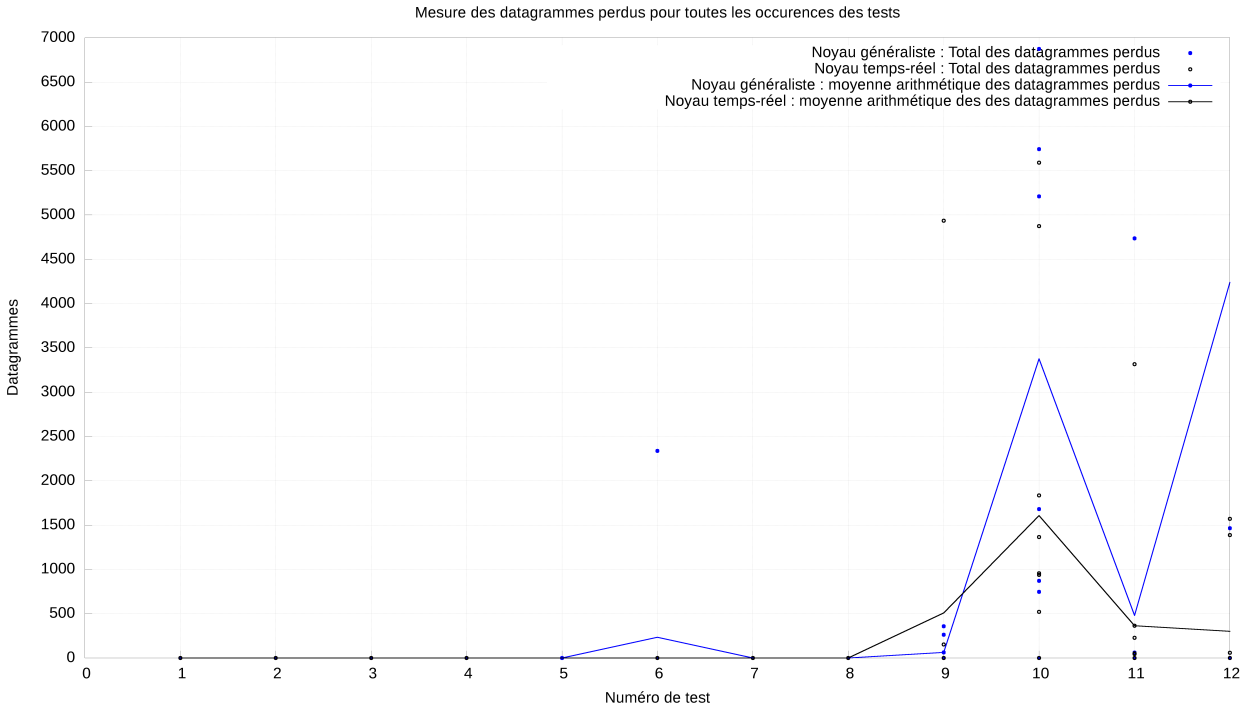

FIGURE 5.24 – Totaux des datagrammes perdus – Noyau généraliste et noyau temps-réel

5.6 Discussion

Nous avons présenté dans les sections précédentes de ce chapitre toutes les données de notre expérience. Nous avons suivi la présentation classique d'un rapport de laboratoire. En conséquence, nous allons effectuer l'étape finale de ce rapport. Nous allons maintenant discuter des résultats présentés dans la section 5.5 page 56. Pour rappel, nous avons défini précisément l'objectif de cette expérience à la section 5.1 page 43. Nous souhaitions déterminer si l'utilisation d'un noyau temps-réel ou d'un noyau généraliste implique des différences significatives dans les mesures réalisées sur un ensemble commun de métriques. Pour un rappel des métriques utilisées, consulter la sous-section 5.2.3 page 47.

L'analyse des débits révèle des différences dans les performances des deux noyaux. D'abord, les performances sont identiques jusqu'au test 3. Ensuite, le noyau généraliste offre un débit supérieur au noyau temps-réel sur les tests 4 à 7. Après, la tendance s'inverse et le noyau temps-réel offre des débits plus importants. Cette tendance est observable sur les tests 8 à 10. Le test 10 illustre la différence de performance la plus importante entre les noyaux. Elle s'explique par la capacité du noyau temps-réel à régir plus rapidement aux événements. Pour un rappel sur le concept de noyau temps-réel, consulter la sous-section 3.2.2 page 36. Enfin, les tests 11 et 12 exposent une saturation des capacités du montage expérimental. Ils montrent un comportement équivalent des noyaux lorsque les capacités de traitement du matériel ont atteint leurs limites. Le tableau 5.1 présente une synthèse du rapport entre le débit du trafic mesuré et le débit spécifié exprimé en pourcentage pour chaque test et pour les deux noyaux.

Numéro de test	Noyau généraliste	Noyau temps-réel	Différence de performance
1	100 %	100 %	0 %
2	100 %	100 %	0 %
3	100 %	100 %	0 %
4	95 %	93,4 %	1,6 %
5	62,5 %	60 %	2,5 %
6	15,41 %	15 %	0,41 %
7	12,66 %	12,22 %	0,44 %
8	14,67 %	15,33 %	0,66 %
9	12,5 %	13 %	0,5 %
10	15 %	24 %	9 %
11	20 %	20 %	0 %
12	2 %	2 %	0 %

TABLE 5.1 – Rapports entre le débit mesuré et le débit spécifié en fonction des noyaux

Sur l'ensemble des tests, la différence de performance entre les noyaux n'est pas significative. Nous avons calculé la moyenne arithmétique des écarts de performance entre les deux noyaux. Elle indique une performance du noyau temps-réel supérieure de 0,43 % à celle du noyau généraliste. Dans la pratique, cette différence de performance n'est pas perceptible sur un ensemble de tests. Comme l'illustre le tableau 5.1, les différences se remarquent à l'échelle des tests. Seul le test 10 indique un écart de 9 % entre les performances des noyaux. Les autres tests illustrent

une différence de performance oscillant entre 0 et 2,5 %. Nos résultats ne correspondent pas à ceux présentés dans le cahier des charges de D-ITG (voir 2.3.2 page 24). On remarque également la différence importante entre le débit du trafic mesuré et le débit spécifié. Les performances ne se dégradent pas progressivement. Elles varient d'un test à l'autre. Les capacités de traitement de Alpa et Bravo ne sont pas limitantes. Le problème est logiciel. Ce phénomène est causé par les mécanismes d'ordonnancement du système d'exploitation. L'ordonnanceur doit assurer le fonctionnement d'un ensemble d'applications. D-ITG n'en est qu'un parmi d'autres. Il ne dispose pas d'une priorité plus élevée. C'est un facteur limitant. Dans un contexte général, la prise en compte de ces différences dépend de la finalité des expériences. L'expérimentateur doit avoir conscience de ces différences. Il lui incombe la décision de les prendre en compte ou de les exclure de l'interprétation des résultats. En conséquence, l'analyse des débits montre qu'un changement de noyau implique une modification du débit dans le trafic généré. Cependant, cette modification du débit n'est pas toujours significative. Sa prise en considération dépend du contexte expérimental.

L'analyse des délais illustre des différences significatives entre les mesures obtenues avec les deux noyaux. Sur l'ensemble des tests, le délai moyen oscille de 1 à 18 millièmes de seconde pour le noyau généraliste et entre 1 et 4 millièmes de seconde pour le noyau temps-réel. Ainsi, la différence d'estimation varie de 0 à 14 millièmes de seconde selon le noyau utilisé. Ce constat s'explique à nouveau par les capacités du noyau temps-réel à réagir plus rapidement aux événements que le noyau généraliste. Dans le cadre des applications orientée *"temps critique"*, l'estimation du délai de transmission joue un rôle important. Prenons un exemple inspiré de [108]. Soit la gestion de l'électronique dans une voiture moderne. La complexité des capteurs et des calculateurs embarqués peut représenter jusqu'à 2500 signaux échangés entres les différents composants électroniques. Réaliser une connexion filaire un à un entre chaque composant est impossible pour des raisons de coûts, de complexité et d'encombrement. Ainsi, la solution réside dans l'utilisation d'un réseau local embarqué. Des fonctionnalités telles que le freinage sont orientées *"temps critique"*. Il est nécessaire de connaître le délais de transmission entre les composants responsables du freinage du véhicule. C'est une question de sécurité : l'interprétation des délais relève d'une importance capitale. Cependant, nous ne pouvons pas privilégier les résultats d'un noyau par rapport à l'autre. En effet, aucun argument ne nous permettrait de justifier ce choix. Dès lors, le choix de considérer l'écart type du délai moyen le plus large ou le plus restreint appartient à l'expérimentateur. Le choix dépend à nouveau du contexte expérimental.

L'analyse des datagrammes corrobore l'analyse des débits. Le graphique 5.23 page 68 a le même aspect général que le graphique 5.7 page 56. En effet, le débit et le nombre de datagrammes transférés sont deux mesures directement proportionnelles. Prenons le cas des tests 11 et 12. Le nombre de datagrammes transférés est en moyenne de 2,25 millions pour une durée de 30 secondes. Ainsi : $\frac{2,25*800}{30} = 60\ Mb/s$. Le test 6 illustre le noyau généraliste subissant une perte de datagrammes. Le noyau temps réel n'en subit pas. Cette perte est contrebalancée par un débit moyen plus élevé que le noyau temps-réel. C'est pourquoi la quantité totale de paquet transférée est similaire pour les deux noyaux. Le test 10 montre le noyau généraliste subissant une perte moyenne de datagrammes plus élevée que le noyau temps réel. De plus, le noyau temps-réel possède un débit supérieur au noyau généraliste. Cela explique les performances supérieures du noyau temps-réel pour ce test. L'analyse de la gigue illustre des différences entre

les noyaux seulement à l'échelle du cent-millième de seconde. Les différences observées ne sont pas significatives. Dans la pratique, elles peuvent être ignorées.

La discussion des résultats n'apporte pas d'argument sur le choix d'un noyau par rapport à l'autre. Cependant, elle met en évidence la possibilité d'influencer les performances d'un même générateur en apportant des modifications sur le système d'exploitation. Les performances de D-ITG dépendent du contexte dans lequel il est exécuté. Cette constatation s'applique à tous les générateurs de l'espace utilisateur et de l'espace noyau. Les performances des générateurs de l'espace physique ne dépendent pas du contexte d'exécution. Ils disposent de leur propres ressources physiques. Pour un rappel sur la notion d'espace, consulter la sous-section 2.1.3.2 page 11. Nous avons illustré que changer de noyau peut influencer les mesures. Or, un changement de noyau est une mise à jour possible sur un système d'exploitation. La distribution *"Ubuntu GNU/Linux"* permet ce procédé (voir [145]). Les changements induits dans le trafic généré sont significatifs ou non suivant le contexte expérimental. Ce choix est laissé à l'expérimentateur. C'est une source d'ambiguïté pour l'expérimentateur et la communauté intéressée par les résultats. En effet, il est nécessaire de motiver ses décisions lors de l'analyse des résultats. C'est une source de discussion et de controverse : les choix d'un expérimentateur ne sont pas nécessairement ceux d'un autre.

Pour supprimer ces difficultés, utiliser un générateur de l'espace physique est une solution. Cependant, ces générateurs offrent un panel de fonctionnalités limité. Une autre solution serait d'utiliser un générateur disposant des performances des générateurs de l'espace physique et des fonctionnalités des générateurs des espaces utilisateur et noyau. Ce générateur devrait exprimer sa capacité ou son incapacité à produire un trafic réseau conforme aux paramètres de l'utilisateur. Cela pour toute exécution, peu importe le matériel informatique utilisé. Il devrait pouvoir *"s'auto-configurer"* pour générer le trafic réseau définit par l'utilisateur. Par exemple, adapter automatiquement les paramètres de PS et d'IDT pour atteindre le débit spécifié. Pour un rappel sur les notions de PS et d'IDT, consulter la sous-section 2.3.1.2 page 20. Or, nous n'avons trouvé aucun générateur disposant de cette combinaison de fonctionnalités parmi les vingt-sept de notre analyse. L'annexe A page 91 présente l'ensemble des générateurs que nous avons analysé. Dans le chapitre suivant nous décrirons notre solution à ce problème. Nous ferons la description d'une architecture logicielle correspondant à un tel générateur.

Chapitre 6

Proposition d'une solution

Dans ce chapitre, nous proposons une solution aux problèmes soulevés dans l'expérience du chapitre 5. Nous décrirons l'architecture d'un générateur de trafic inspirée de D-ITG. Nous souhaitons proposer une solution capable de résoudre les problèmes suivants :

— Justification du choix du système informatique utilisé pour la génération de trafic.
— Compatibilité du logiciel avec les différents systèmes informatiques
— Adaptation dynamique aux capacités de traitement du matériel informatique
— Adéquation entre le débit spécifié et le débit du trafic généré
— Optimisation des ressources allouées à la génération de trafic

Notre solution est constituée d'un couple d'application. La première est un générateur de trafic. Elle s'exécute sur un ordinateur spécialement dédié à cet effet. La seconde est une application de gestion et de contrôle. Elle peut s'installer conventionnellement comme toute application. La section 6.1 explique les principes généraux de notre solution. Nous y évoquons les concepts, techniques et outils nécessaires à la création de ces applications. Nous proposons spécifiquement des solutions open-source issues des logiciels libres. Les générateurs les plus populaires disposent d'une interface graphique. Elles offrent une plus grande convivialité qu'une interface en ligne de commande. Nous prendrons cet aspect en compte sans faire de compromis sur la capacité du générateur à s'exécuter sur du matériel aux capacités limitées. En effet, une interface graphique nécessite des ressources plus importantes qu'une interface en ligne de commande. Elles ne sont pas toujours disponibles. C'est pourquoi nous intégrons le concept d'adaptation dynamique dans notre générateur de trafic. Ce concept sera décrit dans l'architecture de notre générateur de trafic à la section 6.2 page 75. Nous proposons une solution basée sur le concept de communauté virtuelle. Nous souhaitons inclure la participation des utilisateurs. L'objectif est de réaliser une étude comparative sur le matériel utilisé dans les expérimentations. Les résultats dégagés sont rendus accessibles à tous par Internet. La sous-section 6.2.2 page 77 explique ce concept et son utilité.

6.1 Principes généraux

Nous avons observé une variabilité dans les résultats expérimentaux du chapitre 5 page 43. Elle est due au système d'exploitation. Ce sont les mécanismes d'ordonnancement qui influencent les résultats expérimentaux. Or, une application ne possède aucun contrôle sur l'or-

donnanceur du système d'exploitation. Une application contient un ensemble de logiciels et de programmes à exécuter mais elle n'est pas responsable de sa propre exécution. En effet, c'est le système d'exploitation qui détermine le programme à exécuter sur le processeur pour un instant donné. Pour un rappel sur le principe de l'ordonnancement, consulter la sous-section 3.2 page 34. En conséquence, un générateur de trafic n'a jamais le contrôle total sur le trafic qu'il génère. Il existe toujours une variabilité induite par le système d'exploitation. Nous voulons réduire ce facteur d'incertitude. Ainsi, nous proposons de fusionner le système d'exploitation et le générateur de trafic. Nous obtenons alors un système d'exploitation minimaliste spécialisé dans la génération de trafic. Tout les composants du système d'exploitation peuvent être spécialisés pour cette tâche. En outre, nous possédons l'ensemble des capacités de traitement de la machine pour la génération de trafic. Nous possédons un contrôle total sur le processus de génération.

Habituellement, pour pouvoir utiliser un système d'exploitation, il faut l'installer. C'est une tâche laborieuse et chronophage. Heureusement, nous pouvons utiliser un système d'exploitation sans devoir l'installer. C'est possible s'il est contenu dans une image amorçable. Une image amorçable est un fichier informatique utilisable pour démarrer un ordinateur. Ce fichier est copiable sur tout type de médias :

— Disques optiques : CD, DVD, Blue-ray.
— Cartes mémoires flash : SD, MMC, MS, CF, etc.
— Disques durs magnétiques et SSD
— Clés USB
— Bandes magnétiques

Un média contenant une image amorçable se nomme média amorçable. Connecter un média amorçable à un ordinateur lors de son démarrage permet de démarrer le système à partir de ce média. Le contenu de l'image amorçable est alors placé en mémoire centrale pour y être exécuté par le processeur. L'application *"CloneZilla"* utilise ce procédé (voir 5.2.4.6 page 54). Les différentes distributions Linux utilisent aussi ce procédé. Il est connu sous le nom de *"live-cd"*. Il s'agit d'une image amorçable permettant de charger un système d'exploitation en mémoire centrale sans modifier le contenu du disque dur. Les utilisateurs peuvent ainsi essayer la distribution sans conséquence sur leur système personnel. Nous proposons d'utiliser le concept de "live-cd". Il permet d'utiliser notre générateur de trafic sans devoir l'installer. C'est un bénéfice sur deux points : d'une part un gain de temps ; d'autre part, une manipulation plus simple du générateur de trafic pour l'utilisateur.

La portabilité du générateur de trafic est maximale. Elle ne dépend plus d'un système d'exploitation. Elle dépend uniquement du jeu d'instructions supporté par le processeur et des pilotes. Ainsi, nous avons deux dépendances à satisfaire. Premièrement, le jeu d'instructions désigne l'ensemble des instructions machines qu'un processeur est capable d'effectuer. Il représente l'ensemble des opérations élémentaires réalisables par un système informatique. Notre live-cd doit contenir des instructions machines compréhensibles par le processeur de l'ordinateur cible. Il existe deux architectures principale pour les ordinateurs personnel. D'une part, l'architecture X86 pour les processeurs 32 bits ; d'autre part, l'architecture X86_64 pour les processeurs 64 bits. La seconde architecture étant une extension de la première.

Deuxièmement, un pilote est un micro-code logiciel exécuté par le noyau d'un système d'exploitation pour interagir avec un périphérique. Pour un rappel sur la structure des systèmes d'exploitation, consulter la sous-section 3.1 page 31. Les pilotes sont liés à l'architecture de la machine. Les "live-cd" Linux intègrent un ensemble de pilotes génériques et spécialisés. Ils s'exécutent sur la majorité des ordinateurs. Cette technique permet de réduire l'ensemble des dépendances à satisfaire pour exécuter notre générateur de trafic. Un tel système peut être construit à l'aide du projet *"LFS"* qui est l'acronyme de *"Linux From Scratch"* (voir [146]). Ce projet propose la construction d'un système d'exploitation personnalité à la main et à partir de code source uniquement. Aucun fichier binaire exécutable n'est intégré. Les avantages principaux sont au nombre de trois :

— Taille minimale en mémoire centrale
— Définition " à la carte" des composants du système d'exploitation
— Rapidité d'exécution

Une telle approche de construction permet de connaître en détails l'ensemble des composants du système d'exploitation. C'est impossible avec les systèmes d'exploitation prêts à l'emploi. Un choix consciencieux des composants permettrait de comprendre tous les mécanismes mis en œuvre pour générer du trafic.

6.2 Architecture

L'architecture du générateur est illustrée à la figure 6.1 page 76. Elle se compose de deux applications : d'une part, le générateur de trafic ; d'autre part, une application de gestion. Elles possèdent des fonctionnalités complémentaires et peuvent fonctionner indépendamment l'une de l'autre. La sous-section 6.2.1 page 75 expose les concepts du générateur de trafic. La sous-section 6.2.2 page 77 présente les concepts liés à l'application de gestion. Chaque élément présenté à la figure 6.1 sera décrit dans ces deux sous-sections.

6.2.1 Le générateur de trafic

Le générateur de trafic intègre cinq composants. Nous les décrivons dans cette sous-section. Les logiciel *"Sender"* et *"Receiver"* sont directement inspirés de l'architecture de D-ITG. Nous ajoutons une fonctionnalité supplémentaire au logiciel *"Sender"*. Il doit pouvoir *"s'auto-configurer"* pour générer le trafic réseau définit par l'utilisateur. Par exemple, adapter automatiquement les paramètres de PS et d'IDT pour atteindre le débit spécifié. Le composant *"logger"* du générateur de trafic assure le stockage des données expérimentales. Il réalise l'enregistrement de ces données tout en limitant l'impact de la sauvegarde sur le processus de génération de trafic. Les données sont stockées en vue d'être exportées vers l'application de gestion. Deux procédés sont possibles : d'une part, un transfert manuel à l'aide d'un média de stockage amovible ; d'autre part, un transfert réseau.

Ensuite, le générateur de trafic intègre un composant d'initialisation. Ce module détermine les caractéristiques du matériel. Il réalise un ensemble de tests systématiques pour élaborer une base de connaissances. Cette base de connaissances contient des informations quantitatives sur

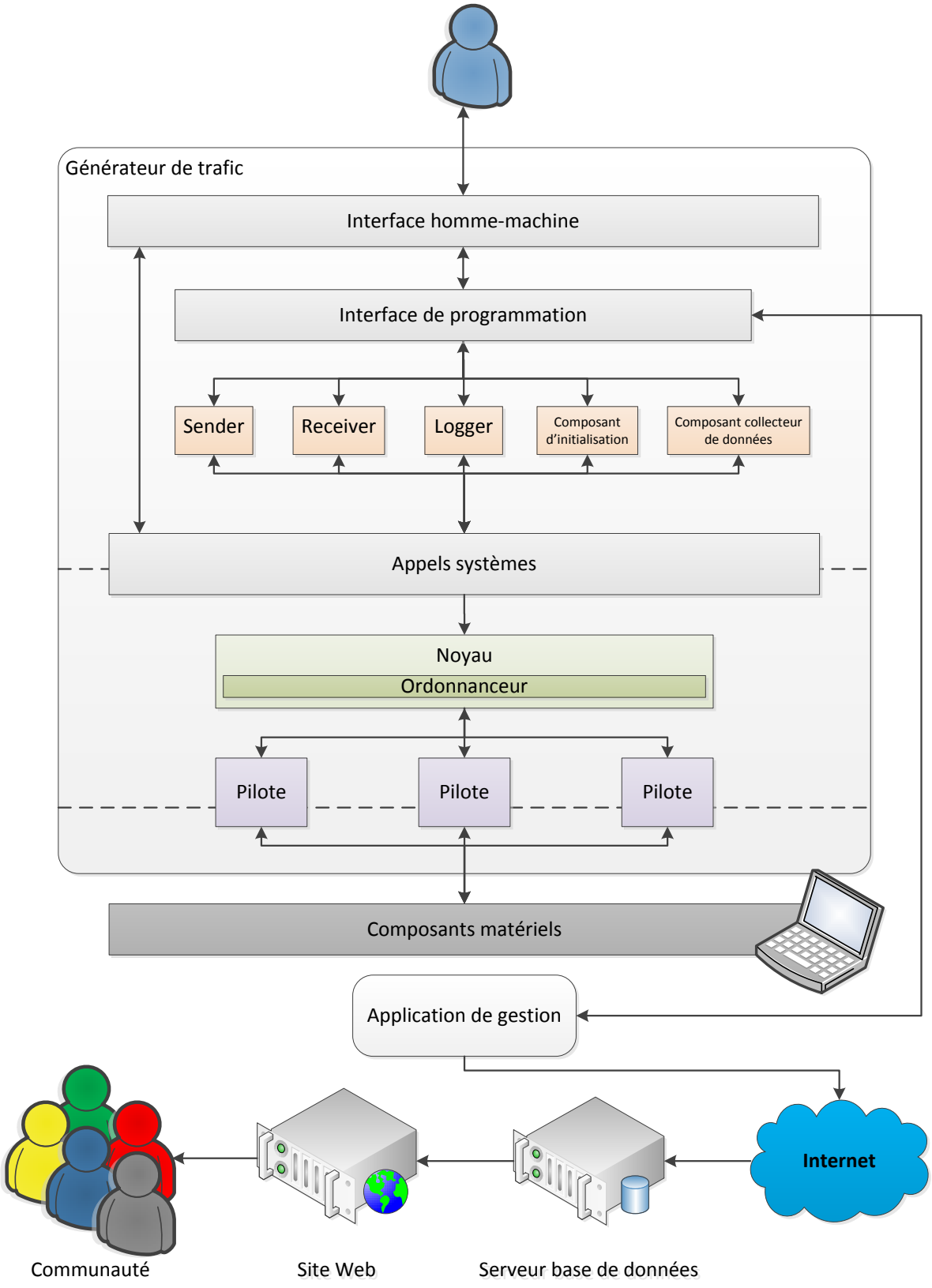

FIGURE 6.1 – Architecture conceptuelle de la solution proposée

les capacités de traitement du système informatique. Elle permet l'adaptation dynamique de l'application. En effet, la base de connaissances sert à sélectionner l'interface graphique la plus adéquate. Cette sélection tient compte des capacités de traitement du système informatique. Dans le domaine des interfaces homme-machine, ce concept est connu sous le nom de plasticité (voir [119]). L'application dispose de trois modes graphiques. Nous les présentons de la plus lourde à la plus légère :

GUI : *"Graphical User Interface"* ou interface graphique utilisateur

TUI : *"Text user Interface"* ou interface utilisateur textuelle

CLI : *"Command Line interface"* ou interface en ligne de commande

Si les capacités de traitement sont trop restreintes, le générateur peut forcer l'utilisation d'un mode graphique plus léger. Sinon, le choix est laissé à l'utilisateur. L'aspect de ces modes graphiques est illustré à la figure 6.2. La base de connaissances possède un second rôle. Elle permet au générateur de trafic d'exprimer sa capacité ou son incapacité à produire un trafic réseau conforme aux paramètres de l'utilisateur. Cette information est une notification à l'utilisateur. Elle n'est pas contraignante. L'utilisateur peut la prendre en compte ou l'ignorer. La capacité s'exprime par rapport aux ressources physiques du système informatique utilisé. Elle ne tient pas compte de la configuration réseau.

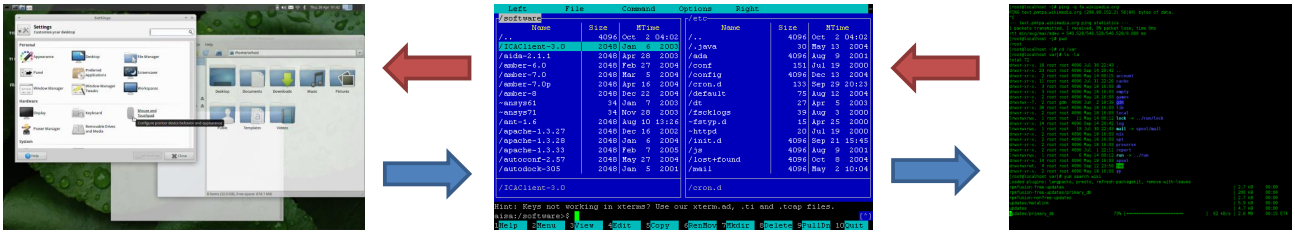

FIGURE 6.2 – Les trois principaux modes graphiques

Enfin, le générateur de trafic intègre un composant collecteur de données. Avec l'accord explicite de l'utilisateur, ce module collecte deux types d'informations : d'une part, la configuration matérielle du système informatique utilisé ; d'autre part, les données calculées par le module d'initialisation. Ces informations sont alors transmises à l'application de gestion. L'application de gestion.

6.2.2 Application de gestion

L'application de gestion est une application conventionnelle. Elle n'a pas besoin de s'exécuter sur un ordinateur dédié. Programmer cette application en langage Java permettrait de la rendre portable sur la plupart des ordinateurs modernes. Elle propose deux fonctionnalités : le contrôle à distance du générateur de trafic et la gestion des données expérimentales. Premièrement, l'application de gestion permet de contrôler à distance une ou plusieurs instances du générateur de trafic. Pour cela, elle utilise l'interface de programmation implémentée par le

générateur de trafic. Nous nous inspirons des fonctionnalités de *"D-ITG Manager"*. Ces fonctionnalités sont décrites à la sous-section 6.2 page 75. Deuxièmement, la gestion des données expérimentales consiste en la collecte et l'export de ces données. L'application de gestion peut recevoir les données expérimentales du générateur de trafic de deux façons différentes. Premièrement, par un média de stockage amovible. Typiquement une clé USB. Deuxièmement, par le réseau. L'application de gestion peut servir à centraliser les données expérimentales une fois qu'elles sont stockées par le logger du générateur de trafic. Le transit des données ne peut pas se faire parallèlement à une génération de trafic. Cela pourrait perturber les expériences. Ensuite, les données expérimentales peuvent être sauvegardées dans un format standard. Lorsque l'application de gestion reçoit les informations du module collecteur de données, elle les transmet à un serveur distant. Elles sont centralisées dans une base de données. L'objectif est de réaliser une étude comparative de ces données. Ainsi, il devient possible de déterminer la configuration matérielle la plus adéquate pour notre générateur de trafic. Plus la quantité d'informations collectée est importante, plus les résultats sont statistiquement significatifs. C'est une recherche d'optimum sur les caractéristiques du matériel utilisé. Une telle étude permettrait également d'identifier les facteurs matériels limitants dans la génération de trafic. Les résultats obtenus sont rendus accessibles à tous par un site internet. Il permet aux utilisateurs de déterminer le matériel à utiliser avec notre générateur de trafic en fonction d'un trafic réseau spécifié. Cette fonctionnalité offre trois avantages :

Minimisation du temps de recherche : le site internet permet d'obtenir des résultats plus rapidement qu'en parcourant la littérature scientifique. Il permet aussi d'éliminer les essais infructueux du type "essai/erreur".

Réutilisation de matériel : du matériel neuf n'est pas toujours nécessaire pour réaliser une expérience. Une configuration standard peut s'avérer suffisante. Ainsi, il est possible de réutiliser du matériel délaissé plutôt que d'acheter du neuf. C'est un choix économiquement plus intéressant.

Justification du matériel utilisé : Le matériel employé pour les expériences est justifié par une étude comparative menée sur l'ensemble des données fournies par les utilisateurs.

Chapitre 7

Conclusion

Au chapitre 2, nous sommes parti d'une vue globale des générateurs de trafic. Nous avons identifié différentes catégories de générateurs pour différents usages. Nous avons alors sélectionné le générateur de trafic D-ITG comme représentant de notre échantillon. Ensuite, nous avons présenté son cahier des charges. Il est constitué de deux parties : d'une part, une présentation générale de son architecture et de ses fonctionnalités ; d'autre par un ensemble de résultats expérimentaux. Nous avons vu qu'il existait de nombreux documents publiés sur cette application. Elles ont rendu l'identification du cahier des charges plus fastidieuse. Nous avons aussi montré des manquements dans le cahier des charges. Les informations qui accompagnent les données expérimentales ne permettent pas de reproduire l'expérience présentée. Ensuite, nous avons étudié les concepts des systèmes d'exploitation dans le chapitre 3. Nous avons présenté leur architecture et les mécanismes de gestion des ressources. Nous avons évoqué le principe général d'ordonnancement et les techniques mise en oeuvre pour y parvenir. Nous avons également présenté la différence entre les systèmes généralistes et les systèmes temps-réel. Nous avons sélectionné le système d'exploitation Debian GNU/Linux pour notre expérience. Enfin, nous avons justifié notre choix par rapport aux besoins de notre expérience. Le chapitre 4 a présenté les technologies réseau les plus répandues et leurs modes de fonctionnement. Nous avons développé les concepts de routeur, switch et hub. Après quoi, nous avons expliqué les modes de transfert de données : simplex, half-duplex et full-duplex. Sur base de ces concepts, nous avons sélectionné un réseau minimaliste pour notre expérience : deux ordinateurs reliés entre eux par un réseau filaire et un switch. Nous avons aussi justifié ce choix.

Le chapitre 5 exposait notre expérience. Nous avons présenté les informations sous la forme d'un rapport de laboratoire en sept étapes. C'est la méthode classique utilisée dans les sciences expérimentales. Elle permet à quiconque de reproduire cette expérience. Nous avons rendus compte des résultats au travers d'une série de 18 graphiques répartis dans quatre sous-sections : l'analyse des débits, l'analyse des délais, l'analyse des datagrammes et l'analyse de la gigue. Pour le débit, les tests illustrent une différence de performance de 0,43 % sur l'ensemble des tests entre un noyau généraliste et un noyau temps réel. Dans la pratique, cet écart n'est pas significatif. Pour percevoir des différences de performances, l'analyse doit se faire à l'échelle des tests. Selon le débit spécifié, la différence de performance entre les noyaux oscille entre 0 et 9 %. L'analyse des débits illustre en plus la différence importante entre le débit spécifié et le débit du trafic mesuré au delà de 80 Kb/s. Le débit généré oscille entre 15 et 95 % de la valeur du débit spécifié. L'expérimentateur doit avoir conscience des écarts de performance. Il

lui incombe la décision de les prendre en compte ou de les ignorer. Les résultats expérimentaux de l'analyse des débits sont corroborés par l'analyse des datagrammes. Elle fournit des données dont les grandeurs sont directement proportionnelles à l'analyse des débits.

L'analyse des délais montre des différences significatives dans les délais OWD calculés par le générateur. Le noyau généraliste offre une moyenne du débit moyen oscillant entre 1 et 14 millièmes de seconde. Le noyau temps réel offre une moyenne du débit moyen oscillant entre 0 et 4 millièmes de seconde. Pour des applications temps critiques tel que le système de freinage d'une voiture moderne, ces évaluations de délais jouent un rôle important. À nouveau, l'utilisateur doit être conscient de ces écarts. La décision lui appartient de les prendre en considération ou de les ignorer. L'analyse de la gigue nécessite d'utiliser une échelle de l'ordre du cent-millième de seconde. Les différences observées ne sont pas significatives. Dans la pratique, elles peuvent être ignorées.

Ces choix d'interprétation laissés à l'expérimentateur sont potentiellement problématiques. C'est une source d'ambiguïté pour l'expérimentateur et la communauté intéressée par les résultats. En effet, il est nécessaire de motiver ses décisions lors de l'analyse des résultats. C'est une source de discussion et de controverse : les choix d'un expérimentateur ne sont pas nécessairement ceux d'un autre. En conséquence, le chapitre 6 proposait une solution pour tenter de remédier à ces écarts de mesure et faciliter la gestion des expériences. Nous avons décrit une nouvelle architecture logicielle. Nous avons proposé de fusionner le système d'exploitation et le générateur de trafic. Soit un live-cd basé sur GNU/Linux spécialisé dans la génération de trafic et accompagné d'une application de gestion. Nous avons aussi détaillé les technologies utilisables pour un tel projet. Ensuite, cette solution intègre un composant d'initialisation permettant deux fonctionnalités nouvelles. D'une part, l'adaptation dynamique du générateur aux capacités de traitement. D'autre part, intégrer la fonctionnalité au générateur d'informer l'utilisateur sur sa capacité ou son incapacité à générer un trafic réseau spécifié. Enfin, l'architecture intègre un composant collecteur de données. Il permet de réaliser une étude comparative des systèmes informatiques utilisés par les utilisateurs. Ce composant permettant à terme de déterminer statistiquement le système informatique le plus performant à utiliser conjointement avec le générateur de trafic. Ces données sont rendues accessibles de tous par un site internet. C'est un projet collaboratif.

Nous terminons par les perspectives d'évolution. Il en existe deux majeures. Premièrement, réaliser une implémentation minimale de la solution évoqué dans ce mémoire. Cette implémentation permettrait de mettre à l'épreuve notre architecture. Deuxièmement, proposer un canevas de spécification standardisé pour le cahier des charges des générateurs de trafic. Il permettrait de faciliter le choix d'un outil en particulier. Il permettrait également d'en cerner facilement les capacités. L'évaluation des performances d'un réseau repose sur le bien-fondé des outils de mesure utilisés. Il est critique de connaître les limites des applications utilisées. Le progrès et les avancées technologiques des réseaux en dépendent directement.

Bibliographie

[1] *IntervalZero RTX and RTX64 product description.*

[2] Luca Abeni, Ashvin Goel, Charles Krasic, Jim Snow, and Jonathan Walpole. A measurement-based analysis of the real-time performance of linux. In *Real-Time and Embedded Technology and Applications Symposium, 2002. Proceedings. Eighth IEEE*, pages 133–142. IEEE, 2002.

[3] Abdolreza Abhari and Mojgan Soraya. Workload generation for youtube. *Multimedia Tools and Applications*, 46(1) :91–118, 2010.

[4] Giuseppe Aceto, Alession Botta, Antonio Pescape, Nick Feamster, M. Faheem Awan, Thahir Ahmad, and Saad Qaisar. Monitoring internet censorship with ubica. In *Traffic Monitoring and Analysis*, volume 9053 of *Computer Communication Networks and Telecommunications*, pages 143–157. Springer international Publishing, April 2015.

[5] Deb Agarwal, José María González, Guojun Jin, and Brian Tierney. An infrastructure for passive network monitoring of application data streams. *Lawrence Berkeley National Laboratory*, 2003.

[6] Akamai. Akamai's state of the internet q1 2015 report. Technical report, Akamai Corporation, 2015.

[7] Ahmed Ait Ali, Fabien Michaut, and Francis Lepage. End-to-end available bandwidth measurement tools : A comparative evaluation of performances. *arXiv preprint arXiv :0706.4004*, 2007.

[8] Gianni Antichi, Andrea Di Pietro, Domenico Ficara, Stefano Giordano, Gregorio Procissi, and Fabio Vitucci. Bruno : A high performance traffic generator for network processor. In *Performance Evaluation of Computer and Telecommunication Systems, 2008. SPECTS 2008. International Symposium on*, pages 526–533. IEEE, 2008.

[9] S. Avallone, S. Guadagno, D. Emma, A. Pescape, and G. Ventre. D-itg distributed internet traffic generator. In *Quantitative Evaluation of Systems, 2004. QEST 2004. Proceedings. First International Conference on the*, pages 316–317, Sept 2004.

[10] Stefano Avallone, Alessio Botta, Alberto Dainotti, Walter de Donato, and Antonio Pescapé. D-itg v. 2.8.1 manual. *http ://traffic.comics.unina.it/software/ITG/manual/*, 2013.

[11] Stefano Avallone, Donato Emma, Antonio Pescapè, and Giorgio Ventre. A distributed multiplatform architecture for traffic generation. In *International Symposium on Performance Evaluation of Computer and Telecommunication Systems, San Jose, CA, USA*, 2004.

[12] Stefano Avallone, Donato Emma, Antonio Pescapè, and Giorgio Ventre. High performance internet traffic generators. *The Journal of Supercomputing*, 35(1) :5–26, 2006.

[13] Stefano Avallone, Antonio Pescape, and Giorgio Ventre. Distributed internet traffic generator (d-itg) : analysis and experimentation over heterogeneous networks, 2003.

[14] Stefano Avallone, Antonio Pescape, and Giorgio Ventre. Distributed internet traffic generator (d-itg) : analysis and experimentation over heterogeneous networks. *poster at ICNP*, 2003.

[15] Victor Yodaiken Micheal Barabanov. A real-time linux.

[16] Jiang Boli. Ixpktgen. *http ://sourceforge.net/projects/ixpktgen/.*

[17] Nicola Bonelli, Stefano Giordano, Gregorio Procissi, and Raffaello Secchi. Brute : A high performance and extensible traffic generator. In *Proc. of SPECTS*, pages 839–845, 2005.

[18] Alessio Botta, Alberto Dainotti, and Antonio Pescapè. Multi-protocol and multi-platform traffic generation and measurement. *INFOCOM*, 2007.

[19] Alessio Botta, Alberto Dainotti, and Antonio Pescapé. Do you trust your software-based traffic generator ? *Communications Magazine, IEEE*, 48(9) :158–165, 2010.

[20] Alessio Botta, Alberto Dainotti, and Antonio Pescapé. A tool for the generation of realistic network workload for emerging networking scenarios. *Computer Networks*, 56(15) :3531–3547, 2012.

[21] Alessio Botta, Alberto Dainotti, and Antonio Pescapé. D-itg site web. *http ://traffic.comics.unina.it/software/ITG/download.php*, 2015.

[22] D.P Bovet and M. Cesati. *Understanding The Linux kernel.* O'Reilly, 2006.

[23] Tim Böttger, Lothar Braun, Olivier Gasser, Felix von Eye, Helmut Reiser, and Georg Carle. Dos amplification attacks – protocol-agnostic detection of service abuse in amplifier networks. In *Traffic Monitoring and Analysis*, volume 9053 of *Computer Communication Networks and Telecommunications*, pages 205–218. Springer international Publishing, April 2015.

[24] Thomas M Chen. "network traffic modelisation" in network in handbook of computer networks. *Hossein Bidgoli*, 2007.

[25] Jean-Noël Colin. Ihdcm035 : Sécurité et fiabilité des systèmes informatiques [présentation diaporamas], chaptire 7. In *IHDCM035.* Faculté d'informatique, 2015.

[26] Intel Corporation. Intel ixp2400 network processor datasheet. Technical report, Intel Corporation, 2004.

[27] Intel Corporation. Intel's next generation integrated graphics architecture –intel® graphics media accelerator x3000 and 3000. Technical report, Intel Corporation, july 2006. Document number : 313343-002.

[28] Intel Corporation. Intel® coretm2 extreme processor x6800 and intel® coretm2 duo desktop processor e6000 and e4000 sequences. Technical report, Intel corporation, OCtober 2007. Document Number : 313278-007.

[29] Intel Corporation. Intel® 82566 gigabit platform lan connect. Technical report, intel Corporation, january 2012.

[30] Intel Corporation. White paper intel ixp2400 network processor : Flexible, high-performance solution for access and edge applications. Technical report, Intel Corporation, 2012.

[31] G Adam Covington, Glen Gibb, John W Lockwood, and Nick Mckeown. A packet generator on the netfpga platform. In *Field Programmable Custom Computing Machines, 2009. FCCM'09. 17th IEEE Symposium on*, pages 235–238. IEEE, 2009.

[32] Jeroen de Best and Roel Merry. *Real-Time Linux for Dummies*. Department of Mechanical Engineering Constrol systems Technology group, P.O. Box 513, WH - 1.126, 5600 MB Eindhoven, the Netherlands, January 2009.

[33] C. Demichelis. Rfc3393 : Ip packet delay variation metric for ip performance metrics (ippm). *https ://tools.ietf.org/html/rfc3393*, 2002.

[34] Sven-Thorsten Dietrich and Daniel Walker. The evolution of real-time linux, 2004.

[35] Ulrich Drepper. What every programmer should know about memory. *Red Hat, Inc*, 11, 2007.

[36] D Emma, A Pescape, and G Ventre. D-itg, distributed internet traffic generator, 2004.

[37] François Espinet, Diana Joumblatt, and Dario Rossi. Method the art of nework troubleshooting : a hands on experimental studiy. In *Traffic Monitoring and Analysis*, volume 9053 of *Computer Communication Networks and Telecommunications*, pages 31–45. Springer international Publishing, April 2015.

[38] Xun Fan, Ethan Katz-Bassett, and John Heidmann. Assessing affinity between users and cdn sites. In *Traffic Monitoring and Analysis*, volume 9053 of *Computer Communication Networks and Telecommunications*, pages 95–110. Springer international Publishing, April 2015.

[39] Wu-chang Feng, Ashvin Goel, Abdelmajid Bezzaz, Wu-chi Feng, and Jonathan Walpole. Tcpivo : a high-performance packet replay engine. In *Proceedings of the ACM SIGCOMM workshop on Models, methods and tools for reproducible network research*, pages 57–64. ACM, 2003.

[40] Pierdomenico Fiadino, Mirko Schiavone, and Pedro Casas. Vivisecting whatwhat un cellular networks : Servers, flows and quality of experience. In *Traffic Monitoring and Analysis*, volume 9053 of *Computer Communication Networks and Telecommunications*, pages 49–63. Springer international Publishing, April 2015.

[41] The Linux Fondation. Archive des noyaux linux. *https ://www.kernel.org/*, 2015.

[42] The Linux Fondation. Noyaux linux 4.0.5 standard. *https ://www.kernel.org/pub/linux/kernel/v4.x/*, 2015.

[43] The Linux Fondation. Noyaux linux 4.0.5 temps réel. *https ://www.kernel.org/pub/linux/kernel/projects/rt/4.0/*, 2015.

[44] Nigel Gamble. The linux kernel preemption project.

[45] Georges and Olivier Gardarin. *Le Client-Serveur*. Imprimerie du lion, second edition, Février 1996. ISBN : 2-212-08876-0.

[46] H. Gerstung, Meinberg, and C. Elliott. Definitions of managed objects for network time protocol version 4 (ntpv4). *Internet Engineering Task Force (IETF)*, 2010. ISSN : 2070-1721.

[47] Glen Gibb, John W Lockwood, Jad Naous, Paul Hartke, and Nick McKeown. Netfpga—an open platform for teaching how to build gigabit-rate network switches and routers. *Education, IEEE Transactions on*, 51(3) :364–369, 2008.

[48] gnu.org. *GNU C library.* trouvé le 20 juin 2015 à : http ://sourceware.org/glibc/wiki/HomePage, 2015.

[49] C. Grimm and G. Schlütermann. *IP Traffic Theory and Performance.* ISBN-13 : 978-3540866732. Springer, 2008.

[50] C.F. Hanes and C.K. Mick. Method simulating data traffic on network in accordance with a client/sewer paradigm, August 8 1995. US Patent 5,440,719.

[51] Richard Hay. Packet generator and analyzers : A retrospective. In *Stanford University Networking Seminar*, 2011.

[52] Benjamin Hesmans, Hoang Tran-Viet, Ramin Sadre, and Olivier Bonaventure. A first look at real multipath tcp traffic. In *Traffic Monitoring and Analysis*, volume 9053 of *Computer Communication Networks and Telecommunications*, pages 233–246. Springer international Publishing, April 2015.

[53] G Horn, Amund Kvalbein, J Blomskøld, and E Nilsen. An empirical comparison of generators for self similar simulated traffic. *Performance Evaluation*, 64(2) :162–190, 2007.

[54] Iannello, Giulio, and et al. "experimental analysis of heterogeneous wirelsss networks". *Springer Berlin Heidelberg*, pages 153–164, 2004.

[55] Keith W. Ross James F. Kurose. *Computer Networking : A top-Down Approach.* Addison-Wesley, sixième edition, 2013. ISBN : 0132856204.

[56] Roelof Jonkman. Netspec : A network performance evaluation and experimentation tool. *http ://www. tisl. ukans. edu/Projects/AAI/products/netspec*, 1995.

[57] Roelof JT Jonkman. *Netspec : Philosophy, design and implementation.* PhD thesis, University of Kansas, 1994.

[58] Ossi Karkulahti and Jussi Kangasharaju. Youtube revisited : On the importance of correct measurement methodology. In *Traffic Monitoring and Analysis*, volume 9053 of *Computer Communication Networks and Telecommunications*, pages 17–30. Springer international Publishing, April 2015.

[59] R Khayari, M Rucker, Axel Lehmann, and Adisa Musovic. Parasyntg : a parameterized synthetic trace generator for representation of www traffic. In *Performance Evaluation of Computer and Telecommunication Systems, 2008. SPECTS 2008. International Symposium on*, pages 317–323. IEEE, 2008.

[60] Samad S Kolahi, Shaneel Narayan, Du DT Nguyen, and Yonathan Sunarto. Performance monitoring of various network traffic generators. In *Computer Modelling and Simulation (UKSim), 2011 UkSim 13th International Conference on*, pages 501–506. IEEE, 2011.

[61] Kolesnikov, Andrey, and Martin Kulas. "load modeling and generation for ip-based networks : a unified approach and tool support". *Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance. Springer Berlin Heidelberg*, pages 96–106, 2010.

[62] Chia-Yu Ku, Ying-Dar Lin, Yuan-Cheng Lai, Pei-Hsuan Li, and K.C.-J. Lin. Real traffic replay over wlan with environment emulation. In *Wireless Communications and Networking Conference (WCNC), 2012 IEEE*, pages 2406–2411, April 2012.

[63] Juha Laine, Sampo Saaristo, and Rui Prior. Rude/ crude. *http ://www.atm.tut.fi/rude.*

[64] Beng Ong Lee, Victor S Frost, and Roelof Jonkman. Netspec 3.0 source models for telnet, ftp, voice, video and www tra c, 1997.

[65] Fredrik Lindahl and Olof Torgersson. mgen-an open source framework for generating clinical documents. *Studies in health technology and informatics*, 116 :107, 2005.

[66] Hiedelberg Germany LNCS Editorial, editor. *Traffic Monitoring and Analysis*. Lecture notes in Computer Science. Springer international Publishing, first edition, February 2015. ISBN 978-3-319-17171-5 ISSN 1611-3349.

[67] John W Lockwood, Nick McKeown, Greg Watson, Glen Gibb, Paul Hartke, Jad Naous, Ramanan Raghuraman, and Jianying Luo. Netfpga–an open platform for gigabit-rate network switching and routing. In *Microelectronic Systems Education, 2007. MSE'07. IEEE International Conference on*, pages 160–161. IEEE, 2007.

[68] Robert Love. *Linux kernel development*. Pearson Education, 4e edition, 2010.

[69] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)*, 10(4) :16, 2010.

[70] John H Maloney, Kylie Peppler, Yasmin Kafai, Mitchel Resnick, and Natalie Rusk. Programming by choice : urban youth learning programming with scratch. *ACM SIGCSE Bulletin*, 40(1) :367–371, 2008.

[71] Paul E McKenney, Dan Y Lee, and Barbara A Denny. Traffic generator software release notes. *SRI International and USC/ISI Postel Center for Experimental Networking*, 2002.

[72] Hassan Metwalley, Stefano Traverso, Marco Mellia, Stanislav Miskovic, and Mario Baldi. The online tracking horde : A view from passive measurements. In *Traffic Monitoring and Analysis*, volume 9053 of *Computer Communication Networks and Telecommunications*, pages 111–125. Springer international Publishing, April 2015.

[73] D. Mills, U. Delaware, J. Martin, J. Burbank, and W. Kasch. Network time protocol version 4 : Protocol and algorithms specification. *Internet Engineering Task Force (IETF)*, 2010. ISSN : 2070-1721.

[74] David L Mills. Internet time synchronization : the network time protocol. *Communications, IEEE Transactions on*, 39(10) :1482–1493, 1991.

[75] Sudhakar Mishra, Shefali Sonavane, and Anil Gupta. Study of traffic generation tools. *nternational Journal of Advanced Research in Computer and Communication Engineering*, 2015.

[76] S. Molnar, P. Megyesi, and G. Szabo. How to validate traffic generator ? In *Proc. of the IEEE International Conference on Communications 2013 : IEEE ICC13- 1st IEEE Workshop on Traffic Identification and Classification for Advacned Network Services and Scenarios (TRICANS)*, juin 2013. Budapest, Hungary.

[77] Tatsuya Mori, Takeru Inoue, Akihiro Shimoda, Kazumichi Sato, Keisuke Ishibashi, and Shigeki Goto. Sfmap : Inferring services over encrypted web flows using dynamical domain name graphs. In *Traffic Monitoring and Analysis*, volume 9053 of *Computer Communication Networks and Telecommunications*, pages 126–139. Springer international Publishing, April 2015.

[78] David Mosberger and Tai Jin. httperf- a tool for measuring web server performance. *Hp research Lab*, pages 30–37, 1999.

[79] The ICSI Networking and Security Group. Traffic generators for internet traffic. *http ://www.icir.org/models/trafficgenerators.html*.

[80] Douglas Niehaus, W Dinkel, and SB House. Effective real-time system implementation with kurt linux. In *Real-Time Linux Workshop*, 1999.

[81] Douglas Niehaus et al. Kansas university real-time (kurt) linux, 2004.

[82] Robert Olsson. pktgen the linux traffic genertor. In *Proceedings of the linux symposium volume two*, 2005.

[83] organisation intergourvernementale de la Convention du mètre. *Le système international d'unité. The international System of Units SI*. Bureau international des poids et mesures, 2006.

[84] Domenico Papaleo and Stefano Salsano. The linux traffic generator. *INFOCOM Department Report*, pages 003–004, 1999.

[85] Marcos Paredes-Farrera, Martin Fleury, and Mohammed Ghanbari. Precision and accuracy of network traffic generators for packet-by-packet traffic analysis. In *Testbeds and Research Infrastructures for the Development of Networks and Communities, 2006. TRIDENTCOM 2006. 2nd International Conference on*, pages 6–pp. IEEE, 2006.

[86] Vern Paxson. End-to-end internet packet dynamics. In *ACM SIGCOMM Computer Communication Review*, volume 27, pages 139–152. ACM, 1997.

[87] Vern Paxson and Sally Floyd. Wide area traffic : The failure of poisson modeling. *IEEE/ACM transactions on Networking*, 3(3) :226–244, 1995.

[88] Andres Perez-Garcia, Christos Siaterlis, and Marcelo Masera. Designing repeatable experiments on an emulab testbed. In Ioannis Tomkos, ChristosJ. Bouras, Georgios Ellinas, Panagiotis Demestichas, and Prasun Sinha, editors, *Broadband Communications, Networks, and Systems*, volume 66 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 28–39. Springer Berlin Heidelberg, 2012.

[89] Sundar Pichai and Linus Upson. Introducing the google chrome os. *The Official Google Blog*, 7, 2009.

[90] Nicolas Pons. *LINUX Configuration de la Sécurité et recompilation du noyau*. TechNote collection. ENI Informatique Technique, 2004. ISBN 2-7460-2464-0.

[91] Nicolas Pons. *LINUX (Fedora, Mandriva, Debian..) Gestion des fichiers*. TechNote collection. ENI Informatique Technique, 2006. ISBN 2-7460-3132-9 ISSN 1767-1590.

[92] Debian Community project. Gnu/linux debian project. *https ://www.debian.org/*, 2015.

[93] NTP : Network Time Protocol. http ://www.ntp.org/. *URL*, 2015.

[94] Guy Pujolle. *Les Réseaux Édition 2008*. Eyrolles, sixième edition, 2008. EAN13 : 9782212117578.

[95] Elias Raftopoulos, Eduard Glatz, Xenophontas Dimitropoulos, and Alberto Dainotti. How dangerous is internet scanning ? a measurement study of the aftermath of an internet-wide scan. In *Traffic Monitoring and Analysis*, volume 9053 of *Computer Communication Networks and Telecommunications*, pages 158–172. Springer international Publishing, April 2015.

[96] M.A Rahman, A. Pakstas, and F.Z Wang. Network modelling and simulation tools. *Simulation Modeling Practices and Theory*, 2009.

[97] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, et al. Scratch : programming for all. *Communications of the ACM*, 52(11) :60–67, 2009.

[98] Jim W Roberts. Traffic theory and the internet. *Communications Magazine, IEEE*, 39(1) :94–99, 2001.

[99] Chloé Rolland, Julien Ridoux, and Bruno Baynat. Litgen, a lightweight traffic generator : Application to p2p and mail wireless traffic. In *Proceedings of the 8th International Conference on Passive and Active Network Measurement*, PAM'07, pages 52–62, Berlin, Heidelberg, 2007. Springer-Verlag.

[100] Chloé Rolland, Julien Ridoux, Bruno Baynat, and Vincent Borrel. Using litgen, a realistic ip traffic model, to evaluate the impact of burstiness on performance. In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, Simutools '08, pages 26 :1–26 :8, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[101] Salvo Rossi and Pierluigi. "joint end-to-end loss-delay hidden markov model for periodic udp traffic over the internet". *Signal processing, IEEE Transactions*, 54.2 :530–541, 2006.

[102] Timothy G Saponas and Roger B Demuth. The distributed irmx operating system : A real-time distributed system. *Mission Critical Operating Systems*, pages 208–231, 1992.

[103] Johann Schlamp, Josef Gustafsson, Matthias Wählisch, Thomas C. Schmidt, and Georg Carle. The abandoned side of the internet : Hijacking internet ressources when domain names expire. In *Traffic Monitoring and Analysis*, volume 9053 of *Computer Communication Networks and Telecommunications*, pages 188–201. Springer international Publishing, April 2015.

[104] Johann Schlamp, Ralph Holtz, Oliver Gasser, Andreas Korsten, Quentin Jacquemart, Georg Carle, and Ernst W. Biersack. Investigating the nature of routing anomalies : Closing in on subprefix hijacking attacks. In *Traffic Monitoring and Analysis*, volume 9053 of *Computer Communication Networks and Telecommunications*, pages 173–187. Springer international Publishing, April 2015.

[105] Laurent Schumacher. Ihdcb 336 : Télécommunication et réseaux [présentation diaporamas], chaptire 1. In *IHDCB336*. Faculté d'informatique, 2014.

[106] Steven Shiau. Clonezilla. *http ://clonezilla.org/*, 2015.

[107] Kwangsik Shin, Jinhyuk Kim, Kangmin Sohn, Changjoon Park, and Sangbang Choi. Online gaming traffic generator for reproducing gamer behavior. In *Entertainment Computing-ICEC 2010*, pages 160–170. Springer, 2010.

[108] Françoise SIMONOT-LION and Nicolas NAVET. *Traité I2C des systèmes temps réel – Volume 2, Ordonnancement, rés eaux, qualité de service, chaptire 10.* Hermes science, 2006. ISBN : 2-7462-1304-4.

[109] K. Sleurs, J Potemans, J. Theunis, D. Li, E Van Lil, and A van de Capelle. Evaluation of network traffic worload scaling techniques. *Computer communication*, 30 :3096–3106, 2007.

[110] ESN Software. Iperf 3.0. *https ://github.com/esnet/iperf.*

[111] Joel Sommers and Paul Barford. Self-configuring network traffic generation. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 68–81. ACM, 2004.

[112] Joel Sommers, Hyungsuk Kim, and Paul Barford. Harpoon : a flow-level traffic generator for router and network tests. In *ACM SIGMETRICS Performance Evaluation Review*, volume 32, pages 392–392. ACM, 2004.

[113] Balaji Srinivasan, Shyamalan Pather, Robert Hill, Furquan Ansari, and Douglas Niehaus. A firm real-time system implementation using commercial off-the-shelf hardware and free software. In *Real-Time Technology and Applications Symposium, 1998. Proceedings. Fourth IEEE*, pages 112–119. IEEE, 1998.

[114] Patrick H. Stakem. Flightlinux : a viable option for spacecraft embedded computers.

[115] Shrutika Suri and Vandna Batra. Comparative study of network monitoring tools. *International Journal of Innovative technology and Exploring Engineering (JITEE)*, 1 :63–65, Août 2012. ISSN : 2278-3075.

[116] Andrew S. Tanenbaum. *Computer Networks*. ISBN-13 : 978-0130661029. Prentice Hall, quatrième edition, 2002.

[117] Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2007.

[118] David Thevenin and Joëlle Coutaz. Adaptation and plasticity of user interfaces. In *Workshop on Adaptive Design of Interactive Multimedia Presentations for Mobile Users*, pages 7–10, 1999.

[119] David Thevenin and Joëlle Coutaz. Plasticity of user interfaces : Framework and research agenda. In *Proceedings of INTERACT*, volume 99, pages 110–117, 1999.

[120] Valentin Thirion, Korian Edeline, and Benoit Donnet. Tracking middleboxes in the mobile world with traceboxandroid. In *Traffic Monitoring and Analysis*, volume 9053 of *Computer Communication Networks and Telecommunications*, pages 79–91. Springer international Publishing, April 2015.

[121] Aaron Turner. Tcp replay. *http ://tcpreplay.synfin.net/*.

[122] Victor Uceda, Miguel Rodriguez, Javier Ramos, José Luis Garcia-Dorado, and Javier Aracil. Selective capping of packet payloads for network analysis and management. In *Traffic Monitoring and Analysis*, volume 9053 of *Computer Communication Networks and Telecommunications*, pages 3–16. Springer international Publishing, April 2015.

[123] URL. Harpoon. *http ://cs.colgate.edu/ jsommers/harpoon/*.

[124] URL. Httperf. *http ://www.hpl.hp.com/research/linux/httperf/*.

[125] URL. Iperf. *http ://iperf.fr/*.

[126] URL. Libcap. *https ://github.com/hpcn-uam/HPCAP*.

[127] URL. libcrafter. *https ://code.google.com/p/libcrafter/*.

[128] URL. Mgen. *http ://manimac.itd.nrl.navy.mil/MGEN/*.

[129] URL. Netperf. *http ://www.netperf.org/netperf/*.

[130] URL. Ntgen. *http ://tochna.technion.ac.il/project/NTGen/html/ntgen.htm*.

[131] URL. Ostinato. *http ://code.google.com/p/ostinato/*.

[132] URL. Ostinato. *http ://code.google.com/p/ostinato/issues/detail ?id=39, 2011.*

[133] URL. Pacgen. *http ://pacgen.sourceforge.net/.*

[134] URL. Packetshell. *http ://playground.sun.com/psh/.*

[135] URL. Scapy. *http ://www.secdev.org/projects/scapy/.*

[136] URL. Seagull. *http ://gull.sourceforge.net/index.html.*

[137] URL. Swing. *http ://cseweb.ucsd.edu/ kvishwanath/Swing/.*

[138] URL. Tcpdump. *http ://www.tcpdump.org/manpages/pcap.3pcap.html.*

[139] URL. Tg traffic generator. *http ://www.postel.org/tg/.*

[140] URL. Ttcp. *http ://linux.die.net/man/1/ttcp http ://ftp.arl.mil/ mike/ttcp.html.*

[141] URL. Udpgen. *http ://www.fokus.fhg.de/usr/sebastian.zander/private/udpgen.*

[142] URL. Posix.1-2008. *http ://pubs.opengroup.org/onlinepubs/9699919799/*, 2008.

[143] URL. Windows ce embedded comact. *http ://www.microsoft.com/windowsembedded/en-us/windows-embedded-compact-2013.aspx*, 2013.

[144] URL. Gnu. *http ://www.gnu.org/gnu/linux-and-gnu.en.html*, 2015.

[145] URL. Gnu/linux ubuntu. *http ://doc.ubuntu-fr.org/kernel*, 2015.

[146] URL. Linux from scratch. *http ://www.linuxfromscratch.org/*, 2015.

[147] URL. Mac os x. *https ://www.apple.com/osx/*, 2015.

[148] URL. Microsoft windows. *https ://www.microsoft.com/en-us/windows*, 2015.

[149] Karima Velásquez and Eric Gamess. A comparative analysis of network benchmarking tools. In *Proceedings of the International Conference on Computer Science and Applications (ICCSA'09–WCECS'09). San Francisco, USA*, 2009.

[150] Giorgio Ventre, Avallone Stefano, and Antonio Pescape. "analysis and experimentation of internet traffic generator.". *proc. of NEW2AN*, pages 70–75, 2004.

[151] Kashi Venkatesh Vishwanath and Amin Vahdat. Realistic and responsive network traffic generation. In *ACM SIGCOMM Computer Communication Review*, volume 36, pages 111–122. ACM, 2006.

[152] Kashi Venkatesh Vishwanath and Amin Vahdat. Swing : Realistic and responsive network traffic generation. *Networking, IEEE/ACM Transactions on*, 17(3) :712–725, 2009.

[153] Michele C Weigle, Prashanth Adurthi, Félix Hernández-Campos, Kevin Jeffay, and F Donelson Smith. Tmix : a tool for generating realistic tcp application workloads in ns-2. *ACM SIGCOMM Computer Communication Review*, 36(3) :65–76, 2006.

[154] Tao Ye, Darryl Veitch, Gianluca Iannaccone, and S Bhattacharya. Divide and conquer : Pc-based packet trace replay at oc-48 speeds. In *Testbeds and Research Infrastructures for the Development of Networks and Communities, 2005. Tridentcom 2005. First International Conference on*, pages 262–271. IEEE, 2005.

[155] Sebastian Zander, David Kennedy, and Grenville Armitage. Kute–a high performance kernel-based udp traffic engine. *CAIA (Center for Advanced Internet Architectures) Technical Report*, 2005.

[156] Liang Zhu, Duane Wessels, Allison Mankin, and John Heidemann. Measuring dane tlsa deployment. In *Traffic Monitoring and Analysis*, volume 9053 of *Computer Communication Networks and Telecommunications*, pages 219–232. Springer international Publishing, April 2015.

[157] Noa Zilberman, Yury Audzevich, G Adam Covington, and Andrew W Moore. Netfpga sume : Toward 100 gbps as research commodity. *IEEE Micro*, (5) :32–41, 2014.

Annexe A

Revue des générateurs de trafic IP

Nous donnons dans cette annexe une description des vingt-sept générateurs de trafic IP considérés pour la rédaction de ce document. Ils correspondent aux outils les plus cités que nous avons rencontré dans notre revue de la littérature. Une connaissance du fonctionnement des réseaux IP est nécessaire pour comprendre ces descriptions. Au besoin, les œuvres [116], [94] et [55] sont des références pour l'étude de tels réseaux. Nous utilisons un même canevas pour donner la description de chaque outil. Nous présentons : les dates de début et de fin du projet, la catégorie, le niveau, l'espace, les systèmes d'exploitation supportés, les protocoles supportés, une description de l'outil et les articles de références.

A.1 Analyse et simulation

A.1.1 TMIX :

Il s'agit d'une extension pour les environnements de simulation NS-2 et NS-3. Cet outil génère des flux TCP et UDP synthétiques dans un environnement de simulation à partir d'une trace réseau. L'outil recrée un modèle de flux pour chaque connexion contenu dans cette trace. Le trafic synthétique produit en sortie est statistiquement similaire au trafic de la trace réseau. Il permet à l'utilisateur de créer automatiquement dans l'environnement de simulation les entités intervenant dans les connexions de la trace réseau.
Durée du projet : 2006 à maintenant
Niveau : Flux
Espace : Utilisateur
Protocoles supportés :
- Couche Application : /
- Couche de transport : TCP, UDP
- Couche réseau : IPv4
- Couche liaison de données : /

Système(s) d'exploitation supporté(s) : Aucun, extension de logiciel.
Références : [76], [153]

A.2 Génération modélisée

A.2.1 NTGen :

Ce logiciel est un module pour le noyau linux. Il permet de générer et d'envoyer des paquets IP sur le réseau. Les paquets envoyés ont une charge de données arbitraire et le taux d'émission de ces paquets est variable. L'utilisateur doit spécifier ces informations. Des trafics TCP et UDP peuvent être générés en même temps. D'abord l'utilisateur entre ses paramètres via une interface graphique s'exécutant dans l'espace utilisateur. Ensuite, le module génère et envoie les paquets. Le logiciel permet aussi d'enregistrer le trafic généré. Enfin, des résultats statistiques sur le trafic réseau peuvent être obtenus avec une analyse postérieure du trafic enregistré.
Durée du projet : 2002 à 2003
Niveau : Flux
Espace : Noyau
Protocoles supportés :
- Couche Application : /
- Couche de transport : UDP, TCP
- Couche réseau : IPv4, IGMP, ICMP
- Couche liaison de données : Ethernet, ARP

Système(s) d'exploitation supporté(s) : Linux, noyau 2.6.
Références : [76], [130]

A.2.2 TG :

Émulateur en ligne de commande. Il est capable de produire et de recevoir du trafic réseau unicast et multicast. TG est contrôlé par un langage de script. Ce langage permet la spécification de plusieurs modes d'exécution, l'utilisation de différents protocoles et la configuration manuelle de paramètres. Ces paramètres sont notamment la taille des paquets et le choix de la distribution mathématique à utiliser pour évaluer la quantité de temps entre la réception de deux paquets. Les distributions implémentées sont : constante, uniforme et exponentielle. Cependant, TG ne peut gérer qu'une seule connexion par instance d'exécution. Enfin, il offre la possibilité d'enregistrer un trafic réseau entrant pour l'analyser postérieurement.
Durée du projet : 2002 à 2008
Niveau : Paquets
Espace : Utilisateur
Protocoles supportés :
- Couche Application : /
- Couche de transport : UDP, TCP
- Couche réseau : IPv4
- Couche liaison de données : /

Système(s) d'exploitation supporté(s) : Linux, FreeBSD, Solaris et SunOS
Références : [139], [76]

A.2.3 HttpPerf :

Outil en ligne de commande permettant de mesurer les performances d'un serveur web. Il offre des fonctionnalités de génération de différents trafics HTTP. Le but du logiciel n'est pas

d'offrir un test de performance spécifique mais bien fournir un outil apte à la réalisation de tests unitaires et généraux. Httperf est capable de produire du trafic réseau avec une intensité variable. Il est donc utile pour définir le débit maximum d'un serveur web. En effet, le débit d'un serveur s'exprime comme le nombre de requêtes traitées par unité de temps. Il offre aussi différentes échelle de vue : requêtes, session d'utilisation mais aussi la durée entre chaque requête De plus, après chaque test effectué, httperf affiche les résultats statistiques correspondants.
Durée du projet : 1998 à maintenant
Niveau : Application
Espace : Utilisateur
Protocoles supportés :
- Couche Application : Http, SSL
- Couche de transport : TCP
- Couche réseau : IPv4, IPv6
- Couche liaison de données : /

Système(s) d'exploitation supporté(s) : HP-UX11, RedHat Entreprise Linux AS, OpenBSD, FreeBSD, Solaris 8, OpenSUSE
Références : [124], [78]

A.2.4 SeaGull :

Outil libre et Open source capable de générer plusieurs types de protocoles. Il a été conçu par la société HP dans le cadre du développement de solution VOIP utilisant le protocole SIP. SIP est un protocole standard d'ouverture de session utilisé dans les télécommunications multimédias. Cet outil permet d'effectuer des tests fonctionnels et d'intégration. Sa particularité est de définir les protocoles dans des fichiers de configuration XML nommés dictionnaire. Cela permet d'implémenter ou de modifier très rapidement un protocole. Les auteurs affirment que la démarche s'effectue en moins de deux heures sans connaissance en programmation. Chaque protocole est définit suivant une série de critères standards éditables au travers d'une interface graphique. De plus, SeaGull intègre le concept de scénarios. Les scénarios permettent d'émuler des interactions réseaux entre plusieurs hôtes suivant un procédé établi à l'avance. Un scénario est un fichier XML dans lequel sont décrits les messages échangés entre des hôtes réseaux. On y indique aussi le comportement à adopter lorsqu'un message non attendu est reçu ou lorsque la vérification d'un paramètre échoue. Ainsi, un scénario peut faire intervenir plusieurs protocoles.
Durée du projet : 2006 à 2012
Niveau : Flux
Espace : Utilisateur
Protocoles supportés :
- Couche Application : SIP
- Couche de transport : UDP, TCP, SSL, TLS, SCCP
- Couche réseau : Ipv4, IPv6
- Couche liaison de données : /

Système(s) d'exploitation supporté(s) : Windows, Linux, SunOS, HP-UX
Références : [136]

A.2.5 NetSpec :

Logiciel utilisable en ligne de commande. Il est conçu pour la configuration de matériel réseau mais aussi pour simplifier l'expérimentation des réseaux. En effet, le logiciel permet de découper les expériences en phases successives suivant des règles précises. Par exemple la phase X précède toujours la phase Y, les phases A et B se déroulent en parallèle, etc. Par phase on entend l'établissement d'une ou plusieurs connexions suivant les caractéristiques définies par l'utilisateur. Protocole, débit, IDT, etc. Pour l'IDT, l'utilisateur doit choisir la distribution mathématique à employer. Plusieurs sont disponibles : Pareto, Poisson, logarithmique, gamma et géométrique. Ce logiciel permet d'effectuer des expériences reproductibles et autorise les connexions en parallèle. Netspec fournit également un cadre générique permettant à l'utilisateur de contrôler à partir d'un seul endroit plusieurs processus situés sur des machines différentes. Le logiciel est composé de plusieurs démons. Un démon est un processus qui s'exécute en tâche de fond sans être sous le contrôle direct de l'utilisateur. Chaque démon est capable de générer ou de recevoir un type de trafic spécifique. Les démons doivent être chargés manuellement par l'utilisateur. La documentation du programme explique ce procédé.
Durée du projet : 1994 à 2002
Niveau : Application
Espace : Utilisateur
Protocoles supportés :
- Couche Application : Telnet, FTP, CBR, HTTP
- Couche de transport : TCP, UDP
- Couche réseau : IPv4
- Couche liaison de données : /

Système(s) d'exploitation supporté(s) : Linux
Références : [57], [56], [64]

A.2.6 PACGen :

Logiciel manipulable en ligne de commande. C'est un générateur de trame Ethernet. La charge utile des paquets IP encapsulés peut être définie par l'utilisateur. Les entêtes UDP, TCP et IP peuvent aussi être éditées. Le logiciel dispose d'un compteur indiquant le nombre de paquets émis et permet de spécifier l'IDT. Des fichiers de configuration contrôlent tous les paramètres dont dépendent les fonctionnalités du logiciel. Ces fichiers peuvent être modifiés par l'utilisateur. PacGen ne dispose pas de documentation en ligne. Ces constatations sont faites suivant l'analyse de son code source.
Durée du projet : 2002 à 2013
Niveau : Flux
Espace : Utilisateur
Protocoles supportés :
- Couche Application : /
- Couche de transport : UDP, TCP
- Couche réseau : IPv4
- Couche liaison de données : Ethernet, ARP

Système(s) d'exploitation supporté(s) : Linux, OpenBSD, FreeBSD
Références : [133]

A.2.7 pktgen :

Outil standard sous linux. Il s'agit d'un générateur de paquets. Il est manipulable en ligne de commande. On l'utilise pour effectuer des mesures de débit et définir la stabilité des équipements du réseau. Il est typiquement employé sur des réseaux à haute vitesse. Son architecture lui permet d'atteindre des débits élevés. Plusieurs instances du logiciel peuvent s'exécuter au même instant. Il est possible de partitionner les ressources physiques de la machine afin que chaque instance bénéficie des mêmes performances. Il y a au maximum un flux de trafic réseau généré par instance du logiciel. Cependant, pktgen nécessite des privilèges pour s'exécuter. Seul le compte "root" peut exécuter ce programme. De plus, le logiciel n'utilise pas la pile protocolaire implémentée par le noyau linux. Il interagit directement avec le pilote de périphérique de la carte réseau. La carte réseau à utiliser est spécifiée dans les paramètres du logiciel. Pktgen offre notemment les fonctionnalités suivantes à l'utilisateur : spécifier le débit, préciser l'intervalle de temps entre l'émission de deux paquets et obtenir des statistiques sur les données émises.
Durée du projet :2001 à maintenant
Niveau : Paquets
Espace : Noyau
Protocoles supportés :
- Couche Application : /
- Couche de transport : TCP, UDP
- Couche réseau : IPv4, IPv6
- Couche liaison de données : /

Système(s) d'exploitation supporté(s) : Linux
Références : [82]

A.2.8 Packet Shell :

Enregistreur et émetteur de trames utilisable en ligne de commande. Il dispose aussi d'utilitaires pour la présentation en console des données collectées sur le réseau. Packet Shell a été conçu et est utilisé pour le développement de protocoles. Il permet de créer, modifier, envoyer et recevoir des trames sur un réseau. Les protocoles supportés sont regroupés dans des bibliothèques chargées au lancement du logiciel. Les commandes acceptées sont rédigées en TCL. TCL est un langage de script inventé par John Ousterhout en 1988. Il peut aussi être utilisé comme outil de surveillance à l'échelle d'un réseau local. Pour l'enregistrement de trafic réseau, le logiciel se base sur la bibliothèque libcap. Il peut ainsi capturer n'importe quelle trame en transit sur le réseau. Enfin, Packet Shell offre la possibilité de modifier une trame de la couche application jusqu'à la couche liaison de données. Ces trames modifiées peuvent être émises également.
Durée du projet : 1998 à maintenant
Niveau : Paquets
Espace : Utilisateur
Protocoles supportés :
- Couche Application : /
- Couche de transport : TCP, UDP, SSL,
- Couche réseau : IPv4, IPv6, ICMPv4, ICMPv6
- Couche liaison de données : Ethernet, ARP, OSPF, STP, CDP

Système(s) d'exploitation supporté(s) : SunOS
Références : [134]

A.2.9 IXPKTGen :

Logiciel manipulable en ligne de commande. Il requiert l'utilisation d'une carte réseau Intel équipée de processeurs IXP2400 ou IXP2800. L'architecture de IXPKTGEN est unique. Elle se compose de deux parties autonomes. La première est l'interface utilisateur, la deuxième est la carte réseau. L'interface utilisateur effectue deux actions. Premièrement elle ouvre une connexion TELNET sur l'adresse IP de la carte réseau. Deuxièmement elle attend les paramètres de l'utilisateur. Ces paramètres sont : le débit de données, l'intervalle de temps entre l'émission de deux trames et le nombre de trames à générer. L'utilisateur doit spécifier si la charge de données des trames est aléatoire ou correspond à un profil de données. Un profil de données est un fichier de configuration crée par l'utilisateur. L'utilisateur peut aussi spécifier une trace réseau à rejouer. Une fonctionnalité permet la relecture des traces réseaux enregistrées avec Packet Shell (voir [134]). La carte réseau est autonome. Sur ses processeurs s'exécute une distribution linux embarquée. Il s'agit de Monta Vista Linux. Ce système d'exploitation est l'interface avec le monde extérieur. C'est avec lui qu'interagit l'interface utilisateur. La génération des trames se fait exclusivement sur la carte réseau. Elle est isolée du monde extérieur. Les commandes prises en charge par l'interface utilisateur sont des macros. Elles sont directement traduites en un ensemble d'opérations élémentaires. Ces opérations sont ensuite envoyées à la carte réseau. IXPKTGEN permet à l'utilisateur de spécifier entièrement la charge de données d'une trame Ethernet. Il est utilisé pour la création de nouveaux protocoles.
Durée du projet : 2011 à 2013
Niveau : Paquets
Espace : Physique
Protocoles supportés :
- Couche Application : /
- Couche de transport : /
- Couche réseau : /
- Couche liaison de données : Ethernet

Système(s) d'exploitation supporté(s) : Linux
Références : [16]

A.3 Génération de montée en charge

A.3.1 Mgen :

Le "Multi-Generateur" abrégé MGEN est un logiciel open source développé par le groupe de recherche " PROTocol Engineering Advanced Networking" abrégé PROTEAN situé au " Naval Research Laboratory" abrégé NRL. MGEN offre la possibilité de réaliser des tests de performance et des mesures en utilisant du trafic UDP et TCP/IP. Il s'agit d'un logiciel composé de deux parties : un récepteur et un émetteur. L'émetteur utilise des fichiers de script pour diriger la génération de trafic. Ces fichiers de script peuvent être utilisés pour émuler du trafic réseau du type unicast ou multicast UDP et TCP/IP. Pour y parvenir, le logiciel utilise plusieurs distributions mathématiques : Constant, exponentiel et "on/off". Le récepteur possède des options

permettant de collecter des données sur le trafic reçu pour l'analyser ensuite. Le récepteur peut également utiliser des fichiers de script pour dynamiquement joindre et quitter des groupes IP multicast. Enfin, MGEN peut être utilisé dans des environnements de simulation comme NS-3. Le logiciel dispose d'un module capable d'interagir avec l'environnement de simulation.
Durée du projet : 2005 à maintenant
Niveau : Paquets
Espace : Utilisateur
Protocoles supportés :
- Couche Application : /
- Couche de transport : UDP, TCP
- Couche réseau : IPv4
- Couche liaison de données : /

Système(s) d'exploitation supporté(s) : Linux, FreeBSD, NetBSD, Solaris, SunOS, SGI, DEC
Références : [76] [128]

A.3.2 Rude/Crude :

Logiciel manipulable en ligne de commande. Il est basé sur le paradigme client/serveur. Le client s'appelle Crude et le serveur Rude. Rude dispose de deux fonctionnalités. Premièrement, émettre le contenu d'une trace réseau. Deuxièmement, émettre un flux de paquets IP avec un débit constant. La charge utilise des paquets IP et la durée d'émission du flux doit être spécifiée par l'utilisateur. Il n'est pas possible de spécifier le nombre de paquets à générer. Crude dispose de deux comportements possibles. Soit enregistrer le trafic entrant dans des fichiers log, soit afficher des résultats statistiques en temps réel sur le trafic entrant.
Durée du projet : 2000 à 2002
Niveau : Flux
Espace : Utilisateur
Protocoles supportés :
- Couche Application : /
- Couche de transport : UDP
- Couche réseau : IPv4
- Couche liaison de données : /

Système(s) d'exploitation supporté(s) : Linux, Solaris, SunOS, FreeBSD
Références : [76] [63]

A.3.3 D-ITG :

Logiciel utilisable en ligne de commande basé sur le paradigme "peer to peer". Il est capable de produire de multiple flux réseaux unidirectionnels partant de plusieurs émetteurs vers plusieurs récepteurs. Le trafic à générer peut être spécifié pour sa durée, le nombre de paquets IP à envoyer et la quantité de données transférée. Plusieurs processus stochastiques sont prédéfinis pour fixer la taille des paquets et la valeur de l'IDT. Les distributions mathématiques suivantes sont disponibles : Constante, uniforme, exponentielle, Pareto, Cauchy, normale, Poisson, Gamma et Weibull. D-ITG intègre aussi des modèles de trafic prédéfinis pour des applications types. Il peut être utilisé pour émuler du trafic synthétique ou pour mesurer des valeurs rela-

tives aux paquets IP : débits de données, débit de paquet, Round Trip Time, gigue et le taux de perte des paquets.Les paramètres du logiciel peuvent être définis soit automatiquement, soit par l'utilisateur, soit par un fichier de configuration. La charge utile de données des paquets IP doit être spécifiée par l'utilisateur ou générée aléatoirement.
Durée du projet : 2003 à maintenant
Niveau : Flux
Espace : Utilisateur
Protocoles supportés :
- Couche Application : Telnet, DNS, Quake3, CounterStrike, VOIP
- Couche de transport : UDP, TCP, DCCP, SCTP
- Couche réseau : IPv4, IPv6
- Couche liaison de données : /

Système(s) d'exploitation supporté(s) : Windows, Linux, MontaVista, Snapgear
Références : [21], [36], [13], [9], [14]

A.3.4 Swing :

Outil manipulable en ligne de commande. Il permet d'enregistrer les interactions entre des applications grâce à un modèle mathématique interne de haut niveau. Partant d'un seul point d'observation sur le réseau, Swing extrait des informations statistiques des données en transit. Ces informations sont des distributions mathématiques. Swing en détermine une pour chacun des trois éléments suivants : les utilisateurs, les applications et le comportement du réseau. Il génère ensuite un trafic synthétique dans un environnement de simulation tenant compte de ces trois distributions. Cet environnement de simulation se nomme ModelNet. Il est spécialement conçu pour simuler des réseaux à grande échelle. Le trafic produit est statistiquement similaire au trafic original.
Durée du projet : 2005 à 2009
Niveau : Flux
Espace : Utilisateur
Protocoles supportés :
- Couche Application : /
- Couche de transport : TCP, UDP
- Couche réseau : IP
- Couche liaison de données : /

Système(s) d'exploitation supporté(s) :
Références : [137], [152], [76]

A.3.5 LitGen :

Émulateur pour les réseaux sans fil de type wifi définis par la norme IEEE 802.11. Ce logiciel est basé sur le paradigme client/serveur. Il utilise un modèle mathématique tenant compte du comportement de l'utilisateur et des applications. Ce modèle repose sur trois distributions mathématiques : Weibull, Pareto et log-normale. LitGen base sa génération de trafic sur le concept de sessions. Une session est un agrégat de connexions TCP. Pendant une session, il y a de multiples échanges entre le client et le serveur. LitGen émule du trafic caractéristique des applications "peer to peer" et des services de messageries. Le trafic généré peut être entièrement

synthétique ou statistiquement similaire à une trace réseau spécifiée en entrée.
Durée du projet : 2005 à 2008
Niveau : Paquets
Espace : Utilisateur
Protocoles supportés :
- Couche Application : /
- Couche de transport : TCP
- Couche réseau : IP
- Couche liaison de données : /

Système(s) d'exploitation supporté(s) :
Références : [100], [99]

A.3.6 Harpoon :

Logiciel utilisable en ligne de commande. Il extrait à partir d'une trace réseau ou de mesures en temps réel un ensemble de paramètres qui définissent les propriétés statistiques du trafic à produire. Les traces prises en compte sont issues de NetFlow. Il s'agit d'une architecture réseau créée par Cisco System permettant d'enregistrer des informations sur les flux IP d'un réseau. Les traces générées ne contiennent d'informations que sur la direction des flux. Elle ne contient pas les données échangées. Partant de ces résultats statistiques, Harpoon génère un trafic réseau synthétique statistiquement similaire au trafic de la trace réseau. Ce logiciel est utilisé pour définir la robustesse d'un réseau face à une montée en charge des données. On l'utilise aussi pour la configuration de matériel réseau.
Durée du projet : 2004 à 2005
Niveau : Flux
Espace : Utilisateur
Protocoles supportés :
- Couche Application : /
- Couche de transport : TCP, UDP
- Couche réseau : IP
- Couche liaison de données : /

Système(s) d'exploitation supporté(s) : FreeBSD, Linux, MacOS, Solaris
Références : [123] [112]

A.3.7 YouTube Workload generator :

Application dotée d'une interface graphique basée sur le paradigme client/serveur. La partie serveur détient des vidéos. Chaque vidéo est disponible en plusieurs qualités. À chaque qualité correspond un fichier sur le serveur. Chaque vidéo possède un indice de popularité. Cet indice est déterminé par une distribution mathématique de type gamma. Le client émule l'action type d'un utilisateur sur YouTube. Il effectue des sessions de visites sur le serveur. Une session est constituée de plusieurs requêtes visant à visionner des vidéos. Ces vidéos sont consultées les unes après les autres suivant leur indice de popularité. Le client peut émuler la présence de plusieurs utilisateurs. Le trafic entre le client et le serveur est totalement générique. L'utilisateur peut spécifier un ensemble de paramètres sur les vidéos et le nombre d'utilisateur. Enfin, une autre fonctionnalité permet d'émuler la présence d'un serveur proxy entre le client et le serveur de

vidéos. Client et serveur donnent des résultats statistiques en temps réel sur le trafic entrant et sortant. Le mécanisme de montée en charge s'effectue en augmentant le nombre d'utilisateurs connectés progressivement.
Durée du projet : 2009 à 2012
Niveau : Application
Espace : Utilisateur
Protocoles supportés :
- Couche Application : Youtube
- Couche de transport : indéterminé
- Couche réseau : indéterminé
- Couche liaison de données : indéterminé

Système(s) d'exploitation supporté(s) : windows
Références : [76], [3]

A.4 Génération de débit maximum

A.4.1 UDPGen :

Outil simple et limité. Il ne propose qu'une seule fonctionnalité : évaluer le taux de paquets IP transmis avec succès. Par perte, le logiciel regroupe deux évènements : soit une erreur sur la somme de contrôle du segment UDP, soit une erreur des couches inférieures à la couche de transport. Le logiciel se base sur le paradigme client/serveur. L'utilisateur dispose de quatre paramètres : la taille des paquets IP, le nombre de paquets IP à envoyer, l'adresse IP et le port de destination. Aucune documentation n'est disponible. Seule l'analyse du code source permet de déduire ses informations.
Durée du projet : 2004 à 2012
Niveau : Flux
Espace : Utilisateur
Protocoles supportés :
- Couche Application : /
- Couche de transport : UDP
- Couche réseau : IPv4
- Couche liaison de données : /

Système(s) d'exploitation supporté(s) : Linux
Références : [141]

A.4.2 Kute :

Logiciel basé au départ sur UDPGen (voir A.4.1). Il conserve le paradigme client/serveur et dispose de fonctions plus évoluées. Kute est composé d'un émetteur et d'un récepteur. Les deux sont des modules prévus pour le noyau linux 2.6. Ils s'exécutent uniquement sur des ordinateurs équipés de processeurs Intel ou AMD contenant un compteur de cycle d'horloge. En effet, le logiciel utilise ce registre processeur pour calculer le débit des données à produire. Kute s'exécute dans l'espace noyau. En conséquence, la partie émettrice du logiciel ne peut être interrompue pendant la génération des données. De plus, le logiciel peut générer jusqu'à quatre flux en même temps mais ces flux doivent avoir la même durée. C'est une limitation induite par

l'espace noyau. Les paramètres pris en compte par l'émetteur sont : port et IP de destination, taille des segments, charge utile des segments, TTL et durée de la transmission. L'objectif de l'émetteur est d'envoyer des paquets IP le plus rapidement possible. Le récepteur réalise deux actions. Premièrement il crée des graphiques sur base des données reçues. Deuxièmement, il enregistre dans un fichier log les informations relatives aux segments reçus. Une analyse de ces fichiers log donne des résultats statistiques sur le trafic généré. Les débits de données atteints par Kute sont élevés. Ils permettent de tester les performances des réseaux à haute vitesse. Les concepteurs de l'outil valident ses performances par une expérience disponible dans [155].
Durée du projet : 2005 à 2011
Niveau : Flux
Espace : Noyau
Protocoles supportés :
- Couche Application : /
- Couche de transport : UDP
- Couche réseau : IPv4
- Couche liaison de données : /

Système(s) d'exploitation supporté(s) : Linux, noyau 2.6 et 2.4
Références : [155], [76]

A.4.3 BRUTE :

Émulateur de trafic UDP. On l'utilise pour évaluer les performances d'un réseau à haute vitesse. Son implémentation est très proche des fonctionnalités offertes pas les noyaux linux des familles 2.4 et 2.6. Il respecte le standard POSIX 1b. Ce logiciel génère du trafic UDP suivant différentes distributions mathématiques dont la liste complète peut être consultée dans [17]. Chacune de ces distributions est implémentée dans un module appelé T-module. Ces T-modules sont rédigés dans un langage de script. Ce langage est décrit dans [17]. Cette architecture logicielle laisse la possibilité à l'utilisateur de créer ses propres modules et offre une grande flexibilité d'utilisation. Les T-modules interagissent avec le logiciel à l'aide d'une interface de programmation. D'autres fonctionnalités pour le calcul de la charge utile des segments et le calcul de la taille totale des trames sont également disponibles.
Durée du projet : 2005 à 2009
Niveau : Paquets
Espace : Utilisateur
Protocoles supportés :
- Couche Application : /
- Couche de transport : UDP
- Couche réseau : IPv4, IPv6
- Couche liaison de données : Ethernet, MAC

Système(s) d'exploitation supporté(s) : Linux
Références : [17] [76]

A.4.4 Bruno :

Il s'agit d'un logiciel assisté matériellement, abrégé LAM. Un LAM possède une architecture hybride. Il est composé de deux parties. L'une logicielle, l'autre matérielle. Logiciel et matériel

collaborent ensemble pour offrir de meilleures performances à l'utilisateur. Bruno est une version modifiée du logiciel Brute (A.4.3). La génération des données à émettre sur le réseau n'est plus en charge du noyau linux. Elle est déplacée vers les processeurs de la carte réseau. Cette carte doit être équipée de processeurs appartenant à la gamme Intel IXP2400 ou IXP2800. Ainsi, la composante logicielle gère les caractéristiques liée aux flux, notamment le calcul de la distribution mathématique. La création et l'envoi des données sur le réseau incombent à la carte réseau. Le dialogue entre ces deux composantes se fait par la mémoire dédiée de la carte réseau. La composante logicielle y écrit les instructions nécessaires. Bruno n'offre pas de fonctionnalités en plus que Brute mais les flux générés sont plus précis et peuvent atteindre des débits plus élevés. Les auteurs appuient la validité de ces informations à l'aide de deux expériences décrites dans [8]. On note la similitude de Bruno avec le projet IXPKTGen (voir A.2.9).
Durée du projet : 2008 à indéterminé
Niveau : Paquets
Espace : Physique
Protocoles supportés :
- Couche Application : /
- Couche de transport : UDP
- Couche réseau : IPv4, IPv6
- Couche liaison de données :Ethernet, MAC

Système(s) d'exploitation supporté(s) : Linux
Références : [76], [8]

A.4.5 iperf :

Logiciel utilisé pour tester le débit de données entre deux hôtes du réseau ou la performance d'un câble réseau. Il est basé sur le paradigme client/serveur. Différents modes de mesure sont possibles : unidirectionnel (client vers serveur), bidirectionnel (client et serveur en interaction) ou inversé (serveur vers client). Plusieurs connexions peuvent être établies entre le client et le serveur. De plus, différentes options permettent à l'utilisateur de modifier des paramètres du contrôle de la congestion des connexions TCP : définir la taille de la fenêtre de réception et spécifier l'algorithme du contrôle de congestion à utiliser. Pour UDP, l'utilisateur peut définir manuellement le débit maximal atteignable. Des options plus générales permettent aussi de définir la quantité de données à transférer entre le client et le serveur ainsi que la durée du transfert de ces données.
Durée du projet : 2010 à maintenant
Niveau : Paquets
Espace : Utilisateur
Protocoles supportés :
- Couche Application : /
- Couche de transport : TCP, UDP, SCTP
- Couche réseau : IPv4, IPv6
- Couche liaison de données : /

Système(s) d'exploitation supporté(s) : Windows, Linux, MacOS, Solaris
Références : [125] [110]

A.4.6 Ostinato :

C'est un générateur de paquets open source et multiplateforme. Il dispose d'une interface graphique. Le logiciel est capable d'éditer la plupart des champs de données intervenant dans les protocoles qu'il prend en charge. Il est utilisé pour des tests fonctionnels et des tests de performance. Ostinato dispose d'une interface de programmation permettant à l'utilisateur de définir ses propres protocoles. Ces protocoles sont définis par un langage de script. Une explication de ce langage est disponible dans [131]. Le logiciel est capable de générer plusieurs types de trafic en même temps. Pour chaque trafic à produire, l'utilisateur peut spécifier : le débit, le nombre de paquets et la durée de transmission. Lors de l'émission de trafic, Ostinato contrôle exclusivement les ports de sorties de ce trafic. Cela évite un parasitage des mesures par une intervention non désirée du système d'exploitation. Les données sont émises suivant un débit constant. Aucune autre distribution mathématique n'est proposée à l'utilisateur. Des résultats statistiques en temps réel sont affichés sur le trafic entrant et sortant. Enfin, Ostinato dispose de fonctionnalités de capture et d'affichage des paquets. Par affichage, on entend l'affichage à l'écran du contenu d'un paquet sous une forme lisible pour un humain. Pour la capture de paquets, ostinato utilise la libraire libcap.
Durée du projet : 2011 à maintenant
Niveau : Paquets
Espace : Utilisateur
Protocoles supportés :
- Couche Application : SIP, HTTP, FTP, RTSP, NNTP
- Couche de transport : TCP, UDP
- Couche réseau : IPv4, IPv6, ICMPv4, ICMPv6, IGMP, MLD, IPtunneling
- Couche liaison de données : ARP, Ethernet

Système(s) d'exploitation supporté(s) : Windows, Linux,FreeBSD, OpenBSD , MacOS
Références : [131], [132], [76]

A.5 Moteurs de relecture

A.5.1 Divide and Conquer :

Logiciel manipulable en ligne de commande. Il génère du trafic réseau à partir d'une trace réseau capturée sur un câble de type OC48. La norme OC48 définit des câbles de fibre optique pouvant atteindre des débits jusque 2400 Mbits/s. Ce débit ne peut être atteint par un ordinateur équipé d'un contrôleur Gigabit Ethernet. Or, il s'agit du standard actuel. Le logiciel offre un moteur de relecture fonctionnant sur des ordinateurs standards. Il divise la génération de trafic entre plusieurs ordinateurs. Cela permet d'atteindre le débit d'un câble OC48. En revanche, le logiciel n'offre pas de fonctionnalités d'enregistrement.
Durée du projet : 2005 à indéterminé
Niveau : Flux
Espace : Utilisateur
Protocoles supportés :
- Couche Application : /
- Couche de transport : /
- Couche réseau : IP

— Couche liaison de données : /
Système(s) d'exploitation supporté(s) : Linux
Références : [76], [154]

A.5.2 TCP Replay :

Logiciel utilisable en ligne de commande. Il s'agit à proprement parler d'un ensemble d'utilitaires. TCP replay est basé sur le paradigme client/serveur. Il permet la manipulation de trace réseau au format libcap. Par manipulation on entend les actions suivantes : enregistrement, édition et relecture d'une trace. Les fonctionnalités d'édition permettent de réécrire les informations de la couche application à la couche liaison de données contenues dans une trace réseau. La relecture permet de spécifier le débit de données pour le trafic généré. TCP replay est conçu pour rejouer des scénarios d'intrusion. Cela permet l'élaboration de système de prévention et de détection. Il est aussi utilisé pour tester et configurer du matériel réseau.
Durée du projet : 2004 à maintenant
Niveau : Flux
Espace : Utilisateur
Protocoles supportés :
— Couche Application : /
— Couche de transport : TCP
— Couche réseau : IPv4, IPv6
— Couche liaison de données : /
Système(s) d'exploitation supporté(s) :
Références : [76], [121]

A.5.3 EAR :

Logiciel utilisable en ligne de commande. Il est spécialement prévu pour les réseaux sans fil correspondant à la norme IEEE 802.11. L'environnement joue un rôle important dans la performance des réseaux sans fil. C'est l'environnement qui définit les altérations subies par les paquets IP. EAR est capable d'émuler un environnement générique suivant trois fonctionnalités. Ces fonctionnalités modifient la manière dont la trace réseau est rejouée. Premièrement, le logiciel ajuste l'intensité avec laquelle sont émises les données. Deuxièmement, une fonctionnalité permet d'ajouter du "bruit" dans le trafic généré. Troisièmement, EAR permet de définir un taux de paquets invalides. Ces trois mécanismes sont définissables par l'utilisateur. Ils permettent de recréer les contraintes liées à un environnement. Le logiciel ne permet pas l'enregistrement ni l'édition de trace réseau. Uniquement la relecture.
Durée du projet : indéterminé à 2012
Niveau : Flux
Espace : Utilisateur
Protocoles supportés :
— Couche Application : /
— Couche de transport : /
— Couche réseau : IP
— Couche liaison de données : IEEE 802.11

Système(s) d'exploitation supporté(s) : Linux
Références : [62], [76]

A.5.4 TCPivo :

Logiciel manipulable en ligne de commande. Il est disponible pour les ordinateurs PC à architecture X86. Il peut générer du trafic à partir d'une trace réseau avec un débit jusqu'aux environs de 200 Mbit/s. TCPivo est donc utilisable pour la génération de trafic sur des câbles optiques de norme OC-3. Cette norme spécifie un débit de données maximal égal à 155,22 Mbit/s. Contrairement à TCP Replay (voir A.5.2), ce logiciel permet la reproductibilité des expériences. C'est ce que démontrent les auteurs dans [88]. Le logiciel reproduit fidèlement le contenu d'une trace réseau à haut débit. Il n'est pas restreint par une exécution dans l'espace utilisateur.
Durée du projet : 2003 à 2012
Niveau : Flux
Espace : Noyau
Protocoles supportés :
- Couche Application : /
- Couche de transport : TCP, UDP
- Couche réseau : IP
- Couche liaison de données : /

Système(s) d'exploitation supporté(s) : Linux, noyau 2.6
Références : [76], [39], [88]

Annexe B

Détails des niveaux

Cette annexe présente une vue séparée de chaque niveau et des outils qui s'y rapportent. Il s'agit d'une vue segmentée du tableau général présenté à la figure 2.1.

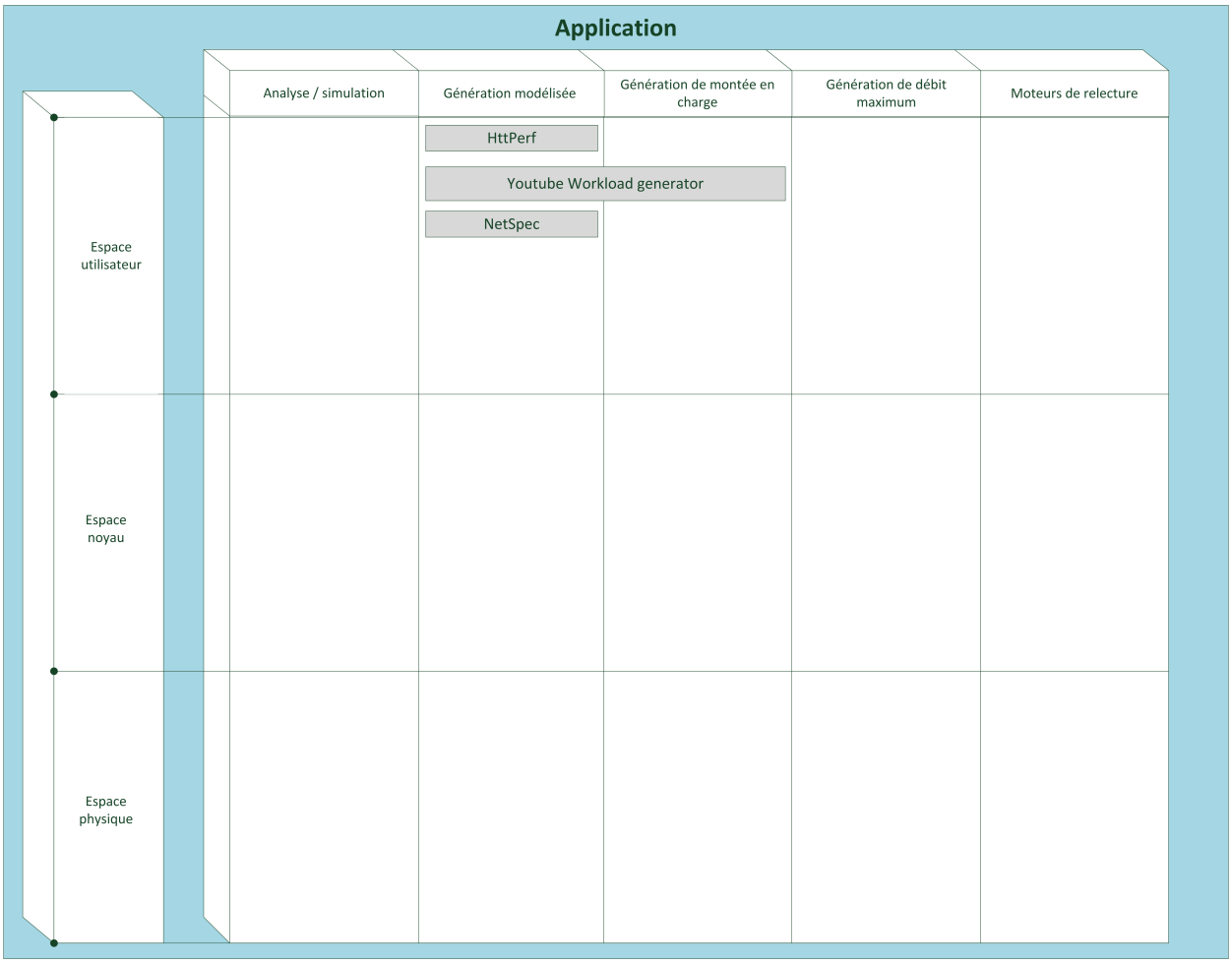

FIGURE B.1 – Vue isolée du niveau application

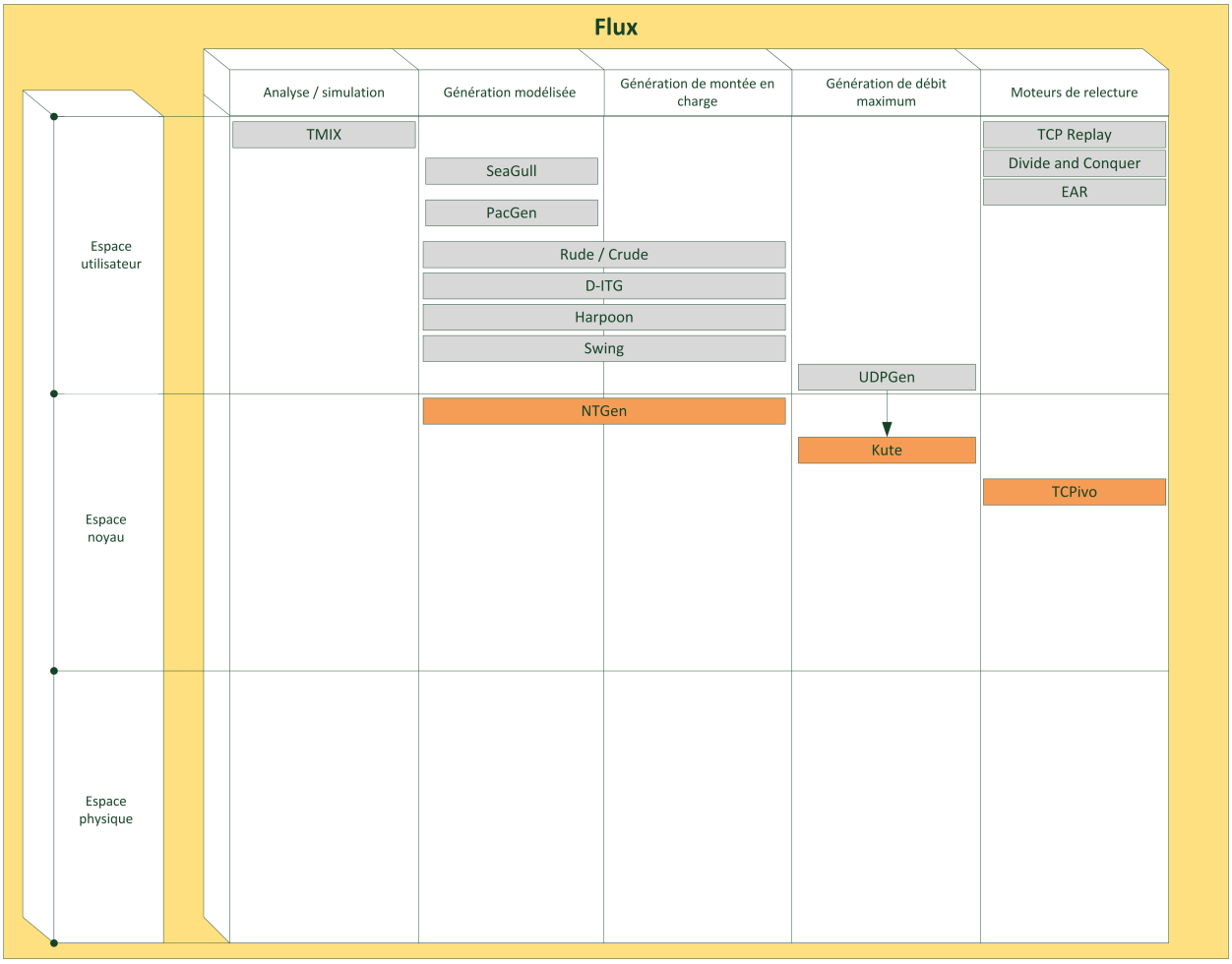

FIGURE B.2 – Vue isolée du niveau flux

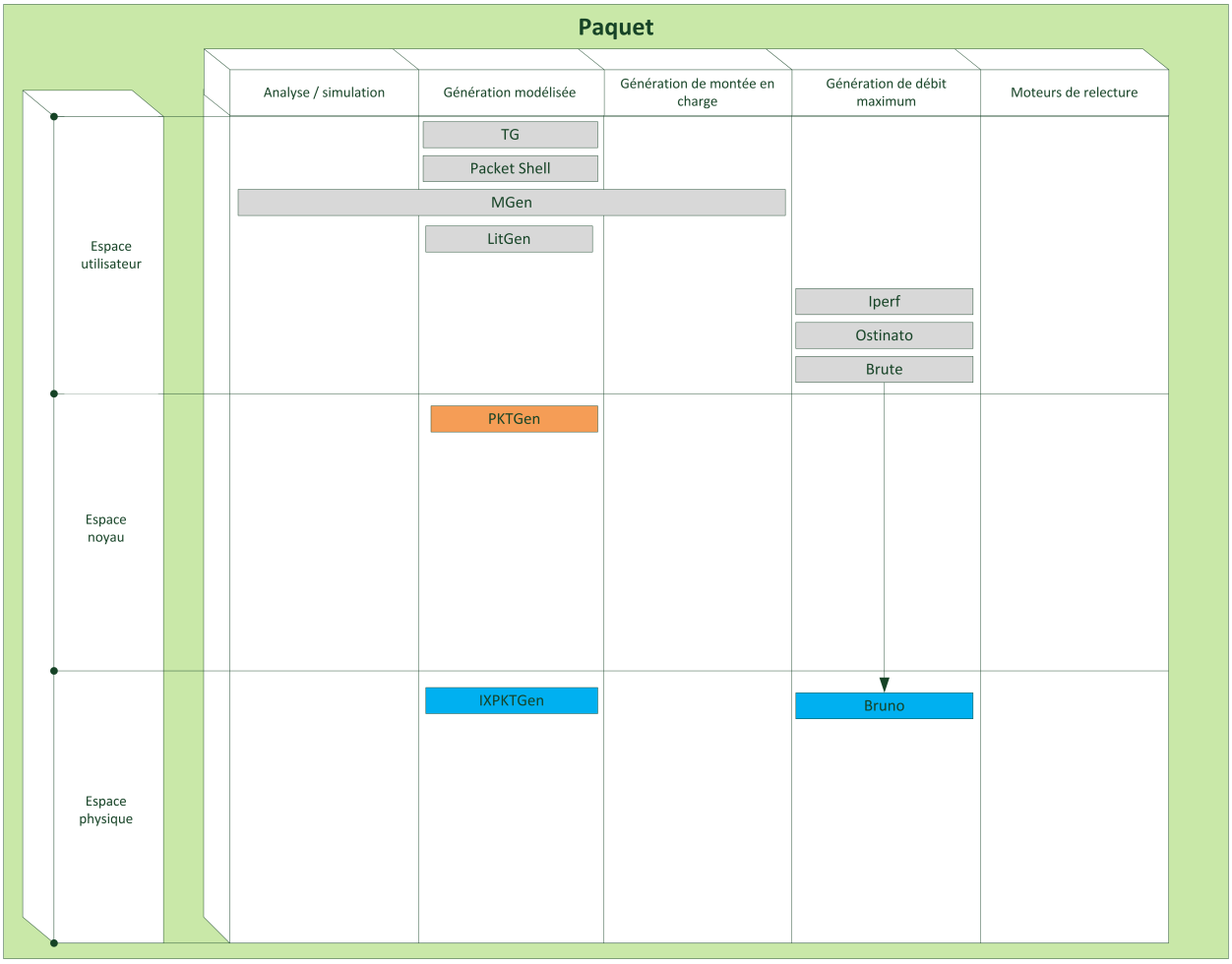

FIGURE B.3 – Vue isolée du niveau paquets

Annexe C

Vues abstraites associées aux niveaux

Ces propos correspondent à une traduction paraphrasée. La paternité de cette section revient à Alessio Botta, Alberto Dainotti et Antonio Pescapé. Les références de cet article sont disponibles en [19]. À la page 160 de cet article l'auteur définit chaque niveau comme une vue abstraite. Intuitivement, c'est la hauteur de vue prise par l'expérimentateur pour analyser un réseau. Les générateurs de trafic travaillent typiquement à trois niveaux d'abstraction. À chaque niveau correspond un certain degré de modélisation. Chaque degré de modélisation définit un cadre de représentation de la réalité et un ensemble de paramètres caractéristiques. Pour les générateurs de trafic cela se traduit par :

Générateurs du niveau applicatif :
Les générateurs de niveau applicatif émulent le comportement d'applications réseaux spécifiques en reproduisant du trafic réseau caractéristique de ces applications.

Générateurs du niveau des flux :
Les générateurs du niveau des flux sont utilisés quand la reproduction d'un trafic réel se fait uniquement au niveau des flux. Les mesures utilisées sont typiquement : le nombre de paquets et de bytes transférés, la durée de transmission du flux, etc.

Générateurs du niveau paquets :
Les générateurs du niveau paquets utilisent des paramètres tels que l'IDT [1] et la taille des paquets. Ces paramètres sont spécifiés par l'utilisateur typiquement en choisissant une distribution statistique pour chacun des deux paramètres.

1. IDT : Inter Departure Time : quantité de temps écoulé entre l'émission de deux paquets.

Annexe D

Fiche technique des ordinateurs

FIGURE D.1 – Ordinateur HP–DC 7700

Audio : Integrated High Definition audio with Realtek ALC262 High Definition audio codec.

Slots d'extension : 2 full-height PCI, 1 full-height PCI Express x1, 1 full-height PCI Express x16.

Mémoire centrale :

- Type mémoire : DDR2-SDRAM PC2-5300 (667MHz).
- Capacité mémoire : 4 GB DDR2 PC2-5300 non-ECC (4 X 1GB).
- Capacité maximale : 4 GB DDR2-Synch DRAM.
- Type et nombre de slots mémoire : DIMM X4.

Réseau :

- Contrôleur réseau : Integrated Intel 82566DM Gigabit.
- Adaptateur réseau : Intel Pro 1000 MT desktop adaptater Gigabit NIC.

Ports :

- 8 x USB 2.0 (avant : 2, arrière : 6).
- 1 standard serial port
- 1 parallel port
- 1 x PS/2 keyboard
- 1 x PS/2 mouse
- 1 RJ-45
- 1 VGA
- 5 Ports audios (avant : casque et micro, arrière : audio-in/audio-out, mic-in).

Alimentation :

- Operating Voltage Range : 90 – 264 VAC
- Rated Voltage Range : 100 – 240 VAC
- Rated Line Frequency : 50/60 Hz
- Operating Line Frequency Range : 47 – 63 Hz
- Rated Input Current : 6.0 A
- Rated Input Current 80 Plus : 5.0 A
- System Heat Dissipation : Typical 307 btu/hr (77 kg-cal/hr) Maximum 1916 btu/hr (483 kg-cal/hr).
- System Heat Dissipation 80 Plus : Typical 239 btu/hr (60 kg-cal/hr), Maximum 1557 btu/hr (392 kg-cal/hr).
- Power Supply Fan : 92 mm variable speed, Power Consumption in ES Mode – Suspend to RAM (S3) (Instantly Available PC) : <3 W.
- Consommation : 365 Watts maximum.
- Voltage en entrée : 90 – 264 / 100 – 240 VAC, 50/60 Hz, 47 – 63 Hz, active PFC.

Graphismes : Intel® Graphics Media Accelerator 3000

Stockage :

- Lecteur optique : 48X Max CD-ROM drive, 48X/32X Combo CD-RW/DVD-ROM drive, 16X/48X DVD-ROM drive, 16X DVD +/-RW DL drive (LightScribe/Double Layer/ Dual Format).
- Disque dur : 80 GB Serial ATA 3.0 Gb/s
- Vitesse de rotation du disque : 7200 RPM
- Type de contrôleur de disque du sous-système : Serial ATA 3.0 Gb/s.

Fréquence du bus mémoire : 667MHz.

Annexe E

Fiche technique du switch

Nous utilisons un switch commercialisé par la société *"PLANET"* sous l'appellation *"SW 501"*. L'ensemble des informations présentées dans cette annexe proviennent de la fiche technique du switch [1]. Les fonctionnalités principales du produit sont :

- Compatibilité avec les standards Fast Ethernet IEEE802.3 et IEEE802.3u
- Mode "store-and-forward"
- Mode Full/Half-duplex sur chaque port, la bande passante totale est de 200 Mb/s par port
- Ajout et apprentissage automatique des adresses sources
- Contrôle du flux des trames IEEEE802.3X PAUSE pour les opérations en mode full duplex
- Diminue les effets du contrôle de flux pour les opérations en mode half-duplex
- Support de 1000 adresses MAC
- Mémoire tampon de 512 KB
- Filtrages Runt et CRC pour éliminer les paquets erronés et optimisé la bande passante du réseau
- Gestion des paquets jusqu'à 1522 bytes
- Gestion du VLAN 802.1Q
- Indicateur LED pour un diagnostic et une gestion plus simple
- MDI/MDIX automatique sur chaque port

Spécifications fonctionnelles :

Switch ASIC : ADMTek ADM6996

Standards : IEEE 802.3 10Base-T Ethernet, IEEE 802.3u 100 Base-TX Fast Ethernet

Protocole : CSMA/CD

Taux de transfert :

- Ethernet : 10 Mb/s (half-duplex), 20 Mb/s (full duplex)
- Fast Ethernet : 100 Mb/s (half-duplex), 200 Mb/s (full duplex)

1. http://www.planet.com.ru/en/product/images/11824/PS-SW501-%20v3_0.pdf

Topologie : étoile

Câbles réseaux :

- 10BASET : 2-pair UTP Cat. 3, 4, 5 (100 m), EIAT/TIA-568 100-ohm STP (100 m)
- 100BASE-TX : 2-pair UTP Cat. 5 (100 m), EIAT/TIA-568 100-ohm STP (100 m)

Nombre de ports : 5 X 10/100 Mb/s

Mode de transmission : Store-and-forward

Taille de la table d'adresses : 1000 entrées

Débit de filtrage et de transfert des paquets :

- 10 Mb/s Ethernet : 14,880 pps
- 100 Mb/s Ethernet : 148,800 pps

Apprentissage des adresses MAC : mise à jour automatique

Spécification physiques :

Front panel

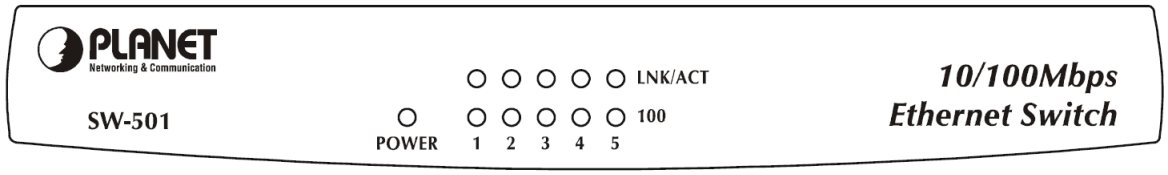

Figure 1. SW-501 front panel

Rear panel

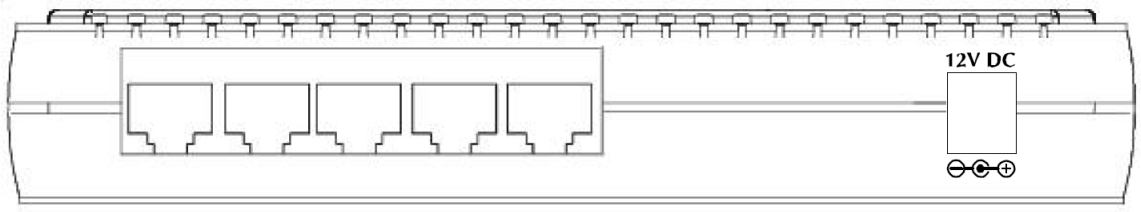

Figure 2. SW-501 Rear panel

FIGURE E.1 – Switch PLANET SW 501

Annexe F

Résultats expérimentaux

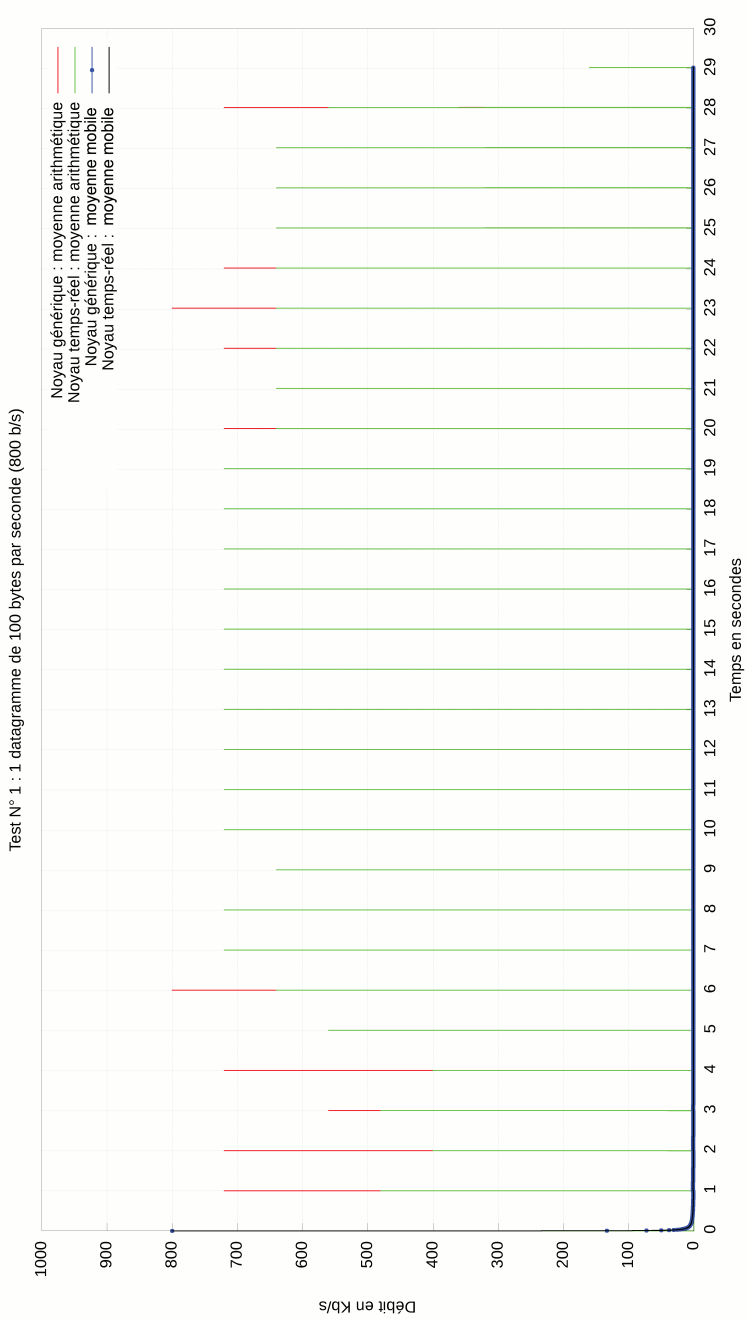

FIGURE F.1 – Analyse des débits – Test 1

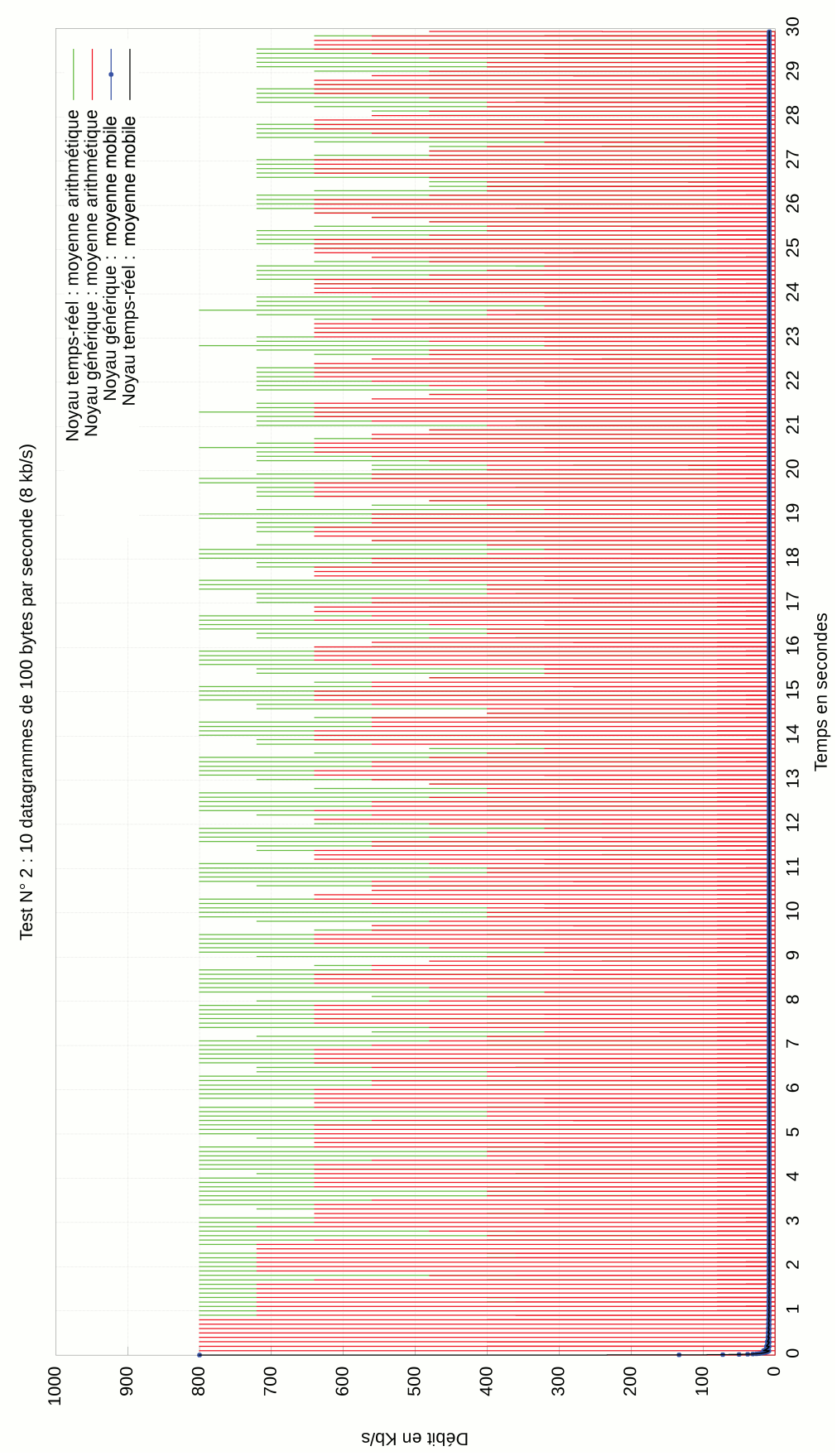

FIGURE F.2 – Analyse des débits – Test 2

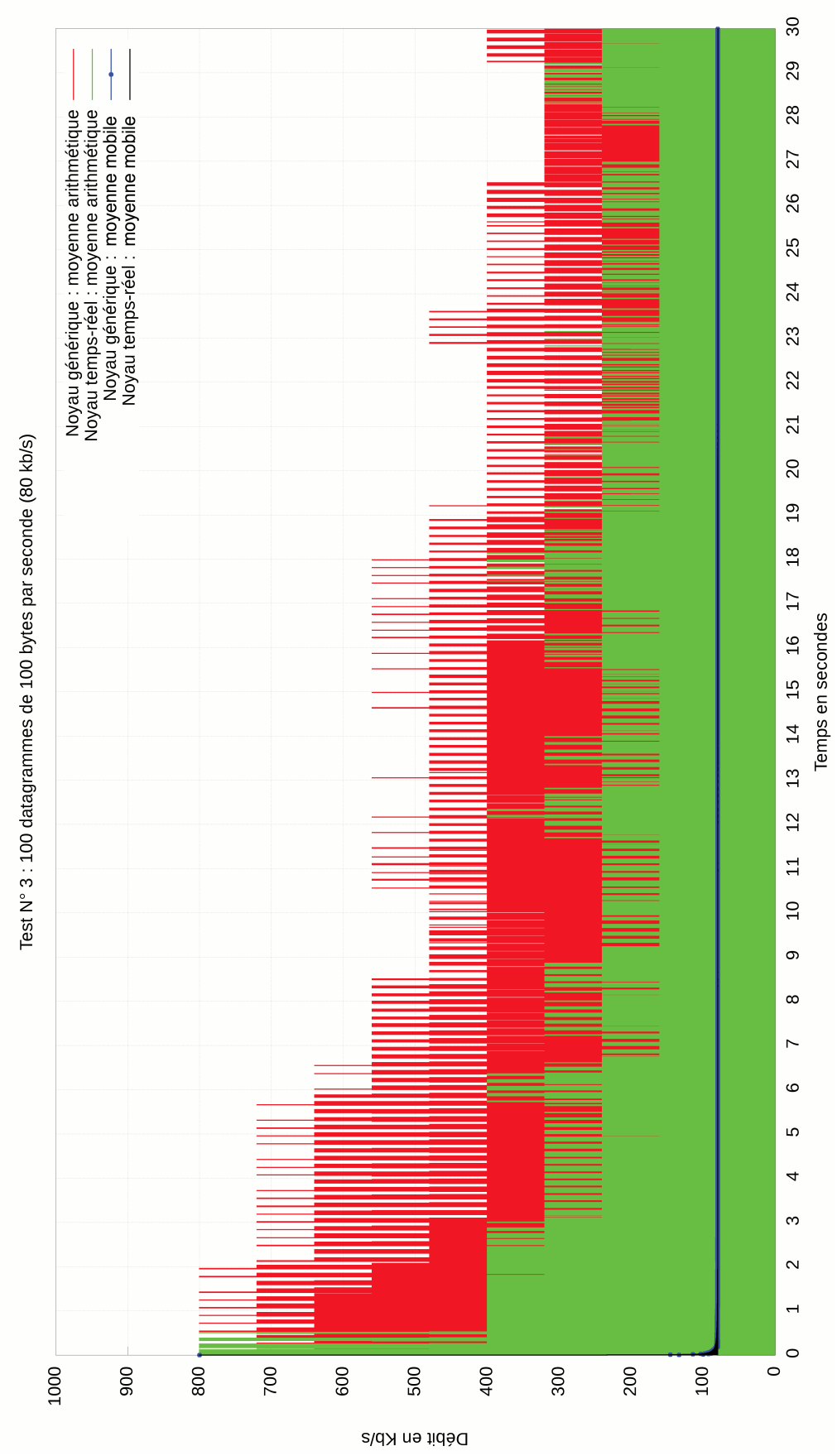

FIGURE F.3 – Analyse des débits – Test 3

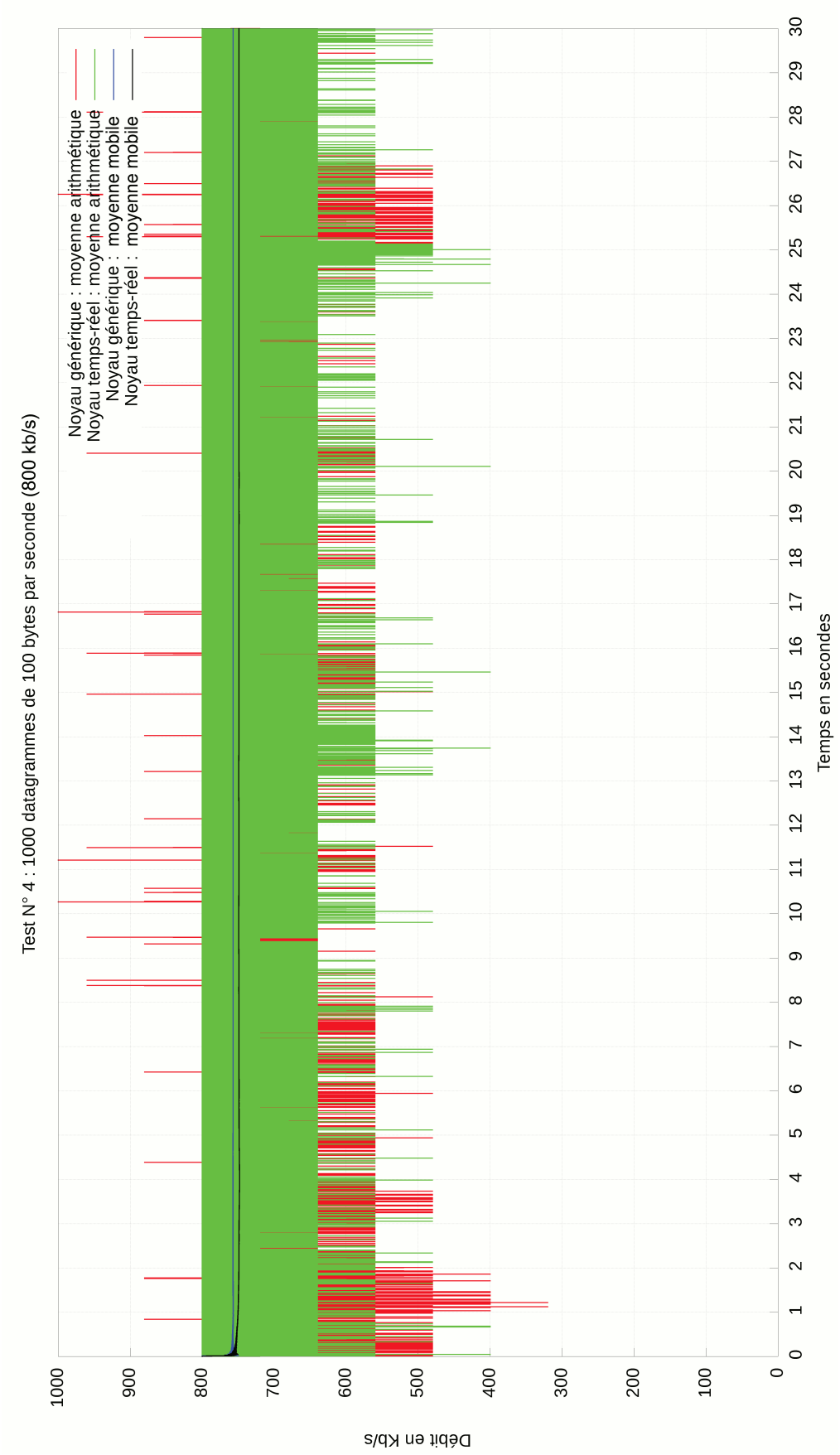

FIGURE F.4 – Analyse des débits – Test 4

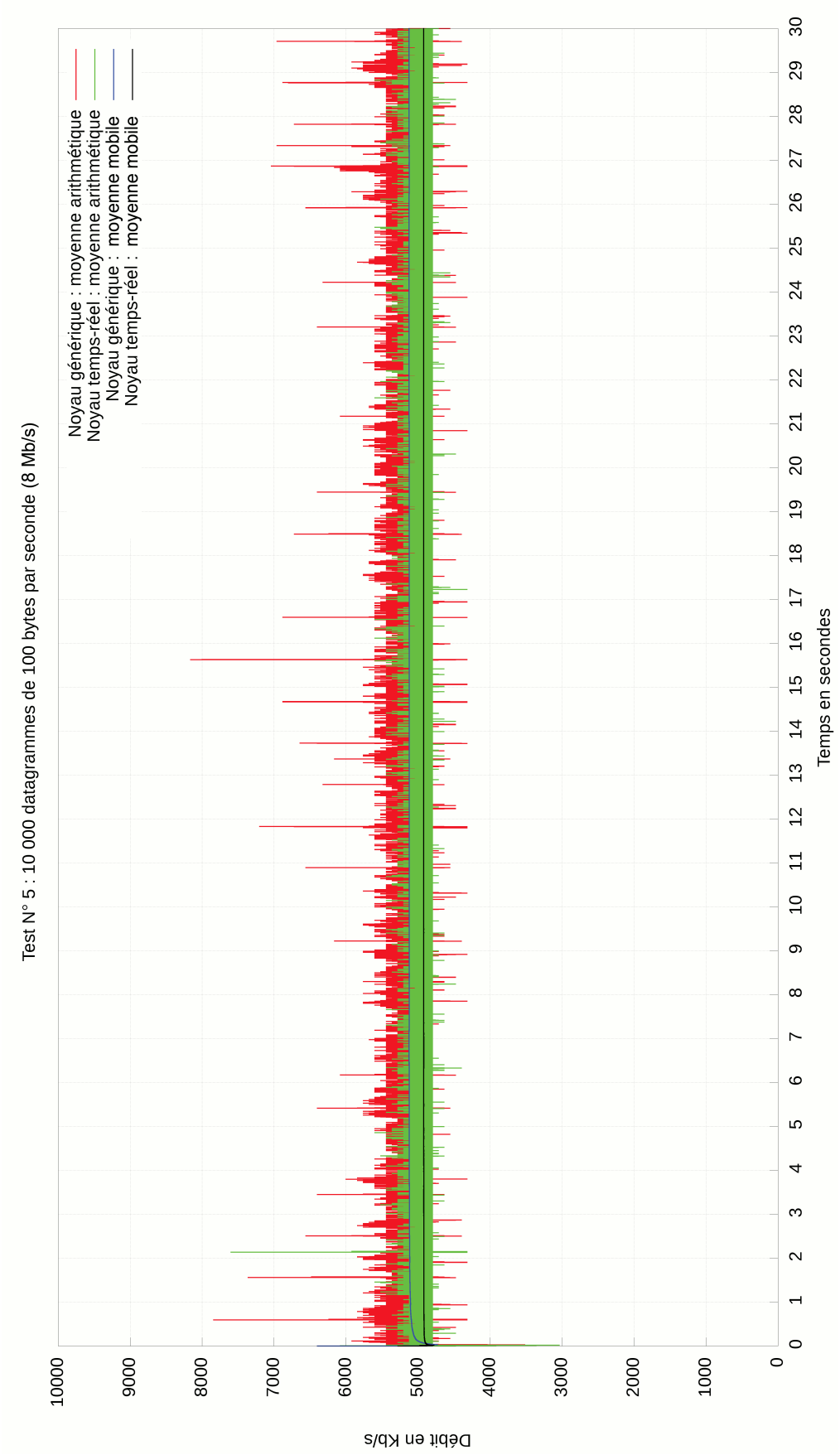

FIGURE F.5 – Analyse des débits – Test 5

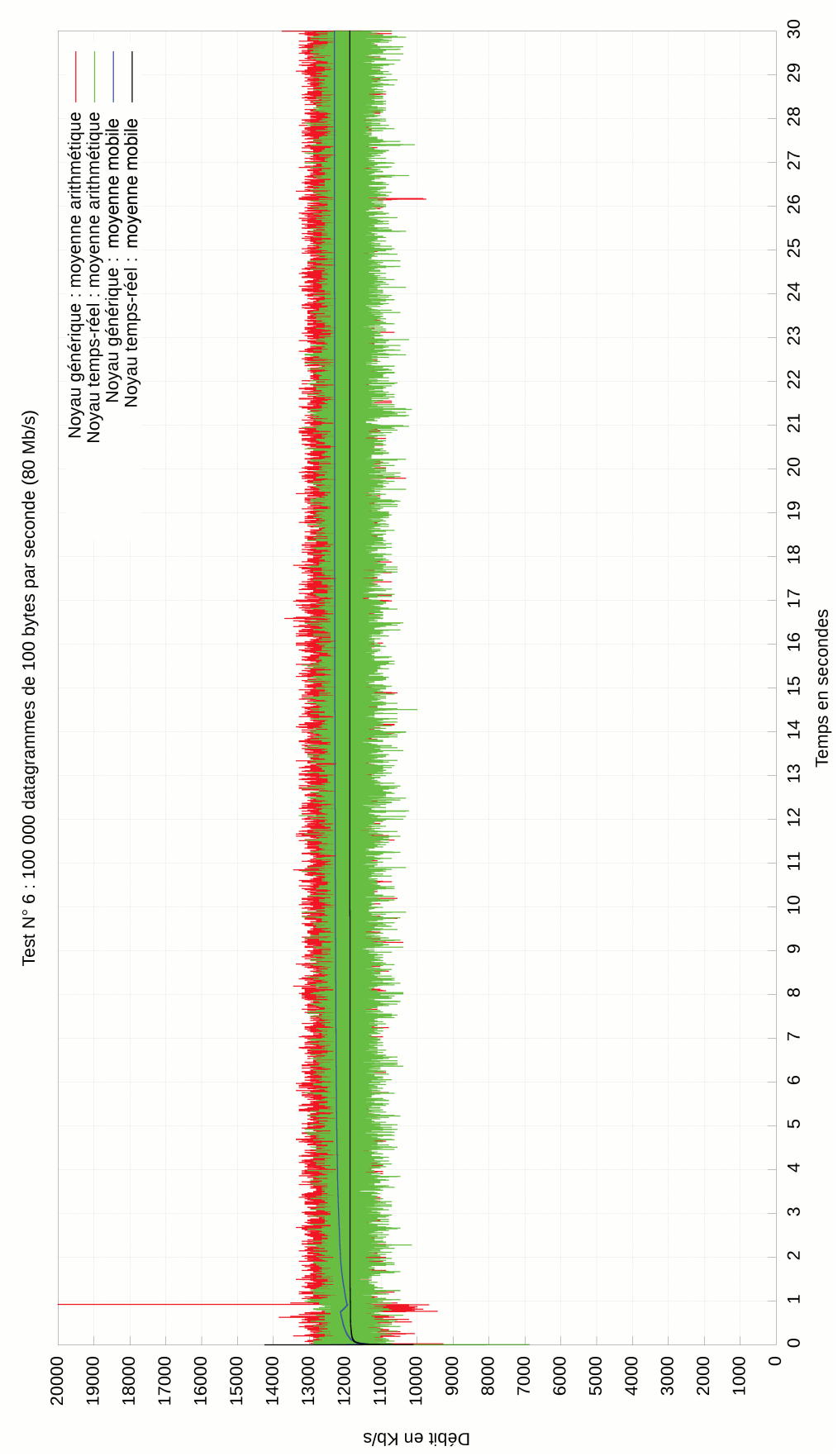

FIGURE F.6 – Analyse des débits – Test 6

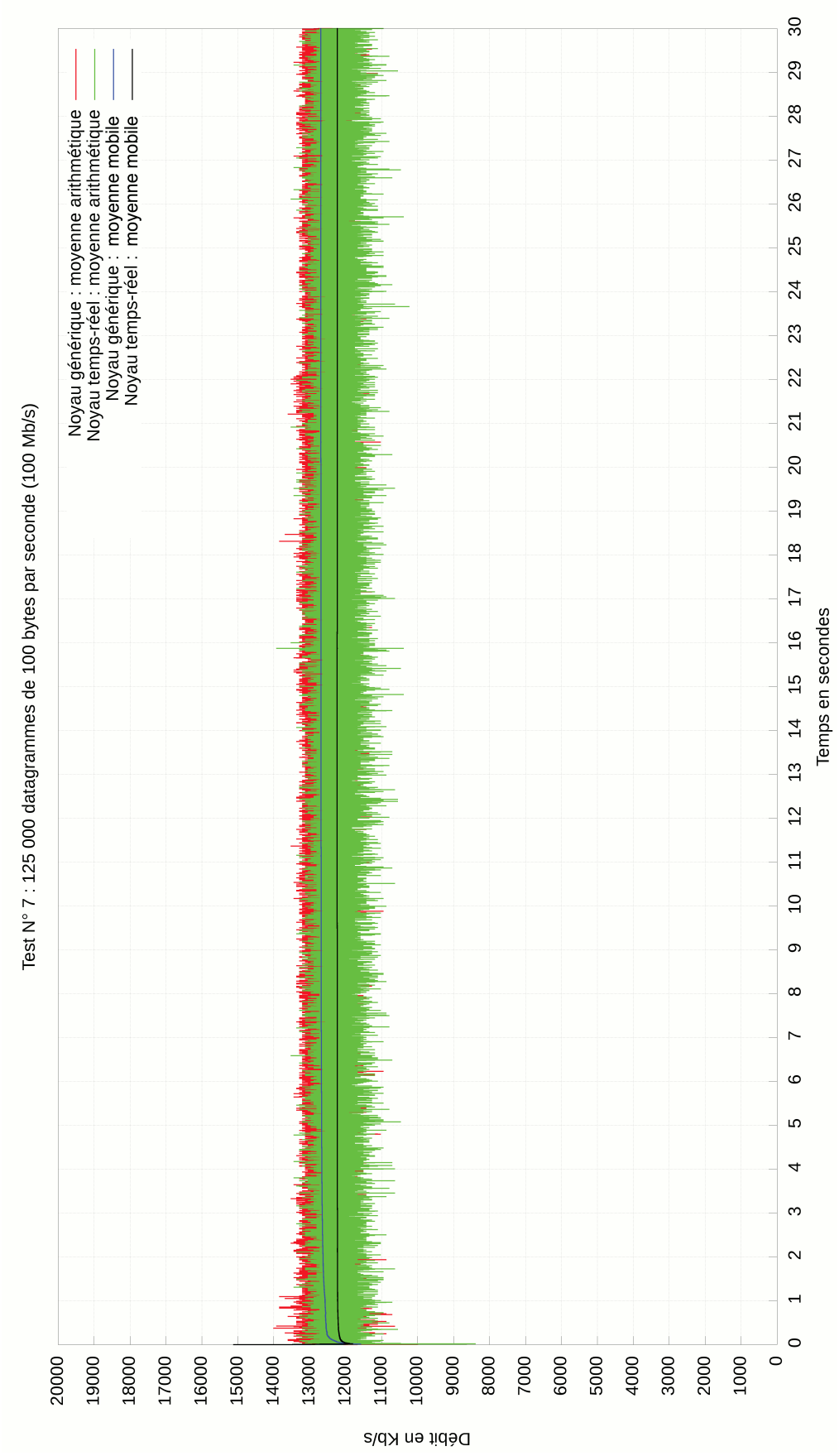

FIGURE F.7 – Analyse des débits – Test 7

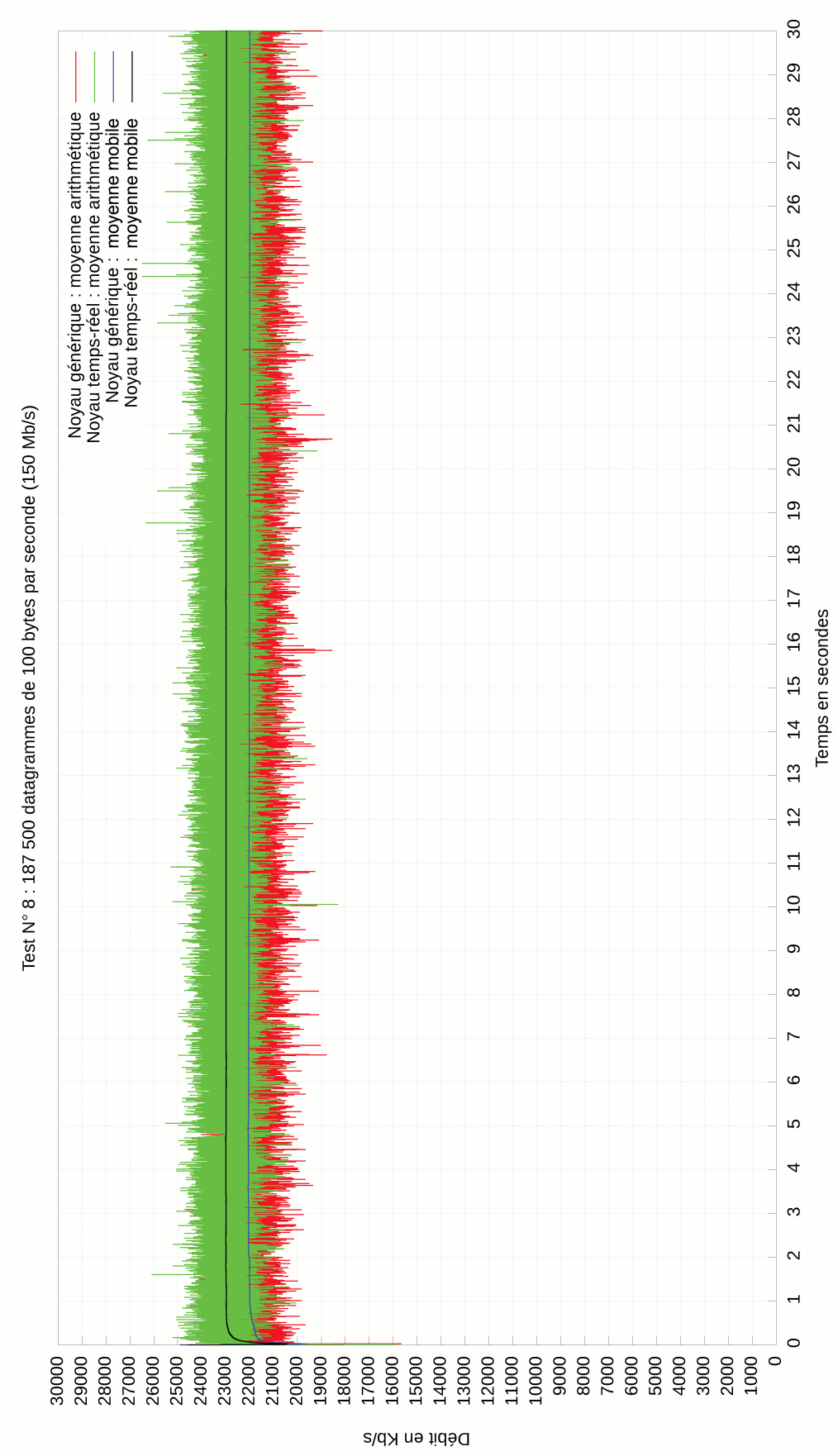

FIGURE F.8 – Analyse des débits – Test 8

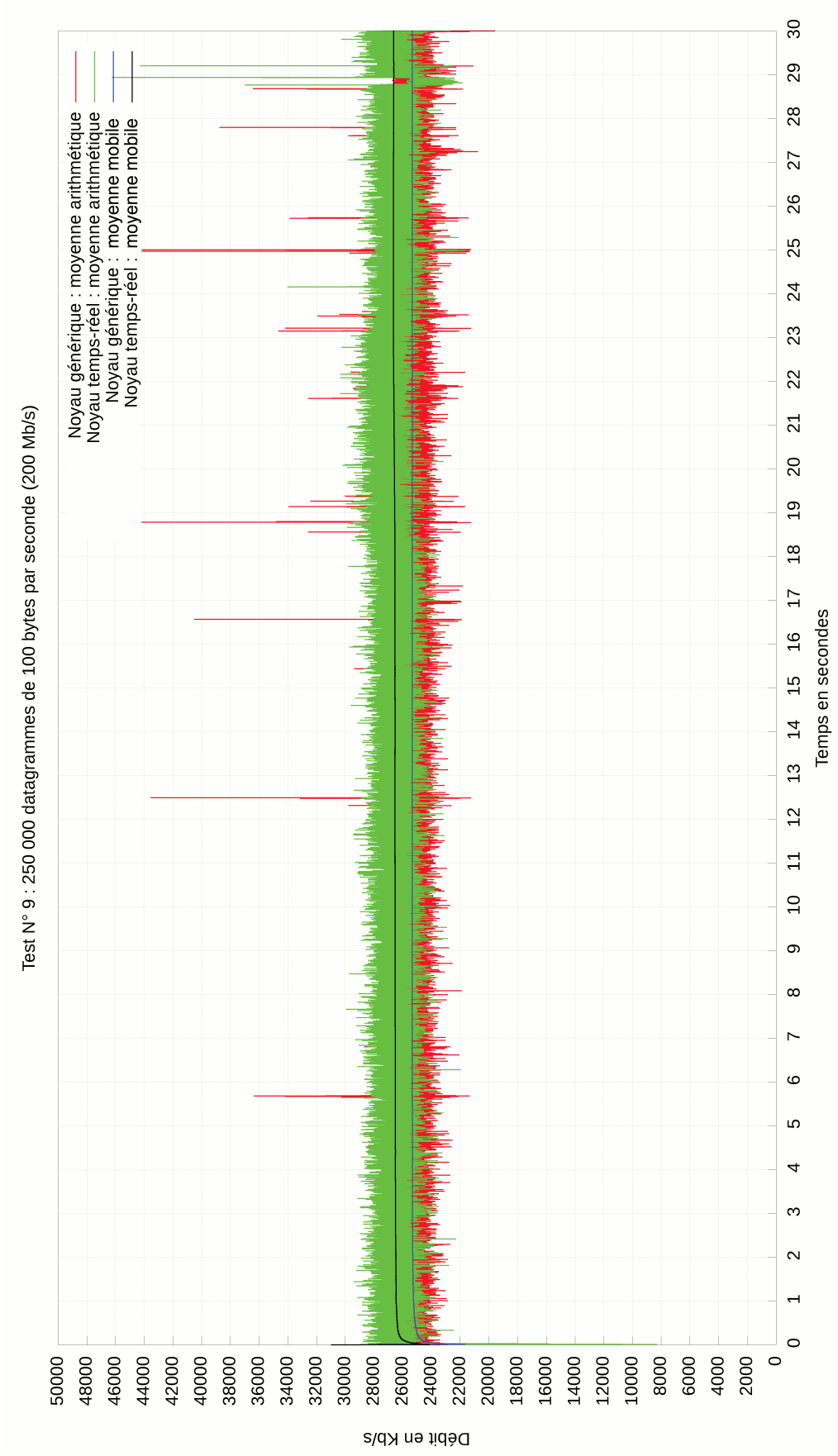

FIGURE F.9 – Analyse des débits – Test 9

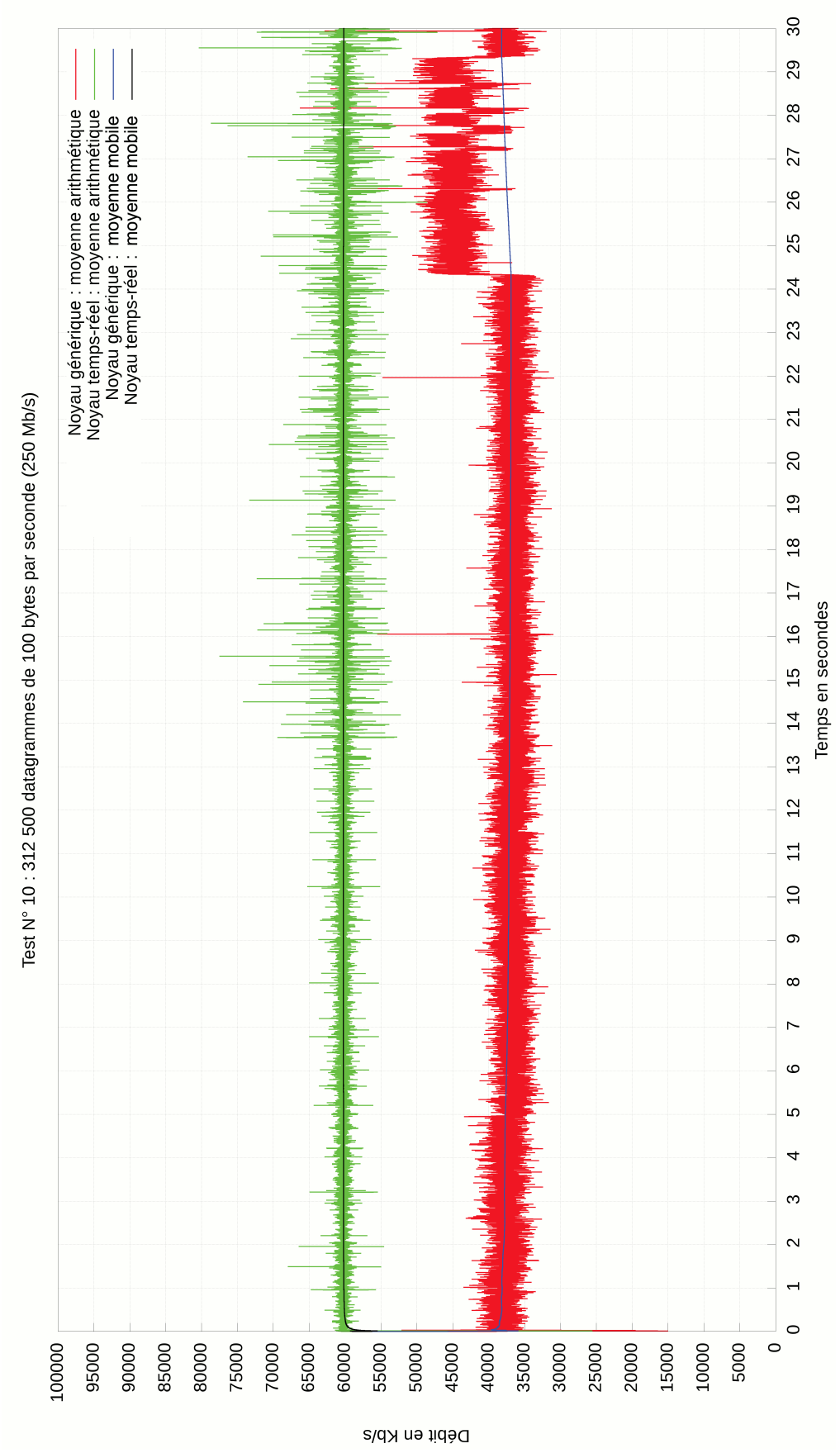

FIGURE F.10 – Analyse des débits – Test 10

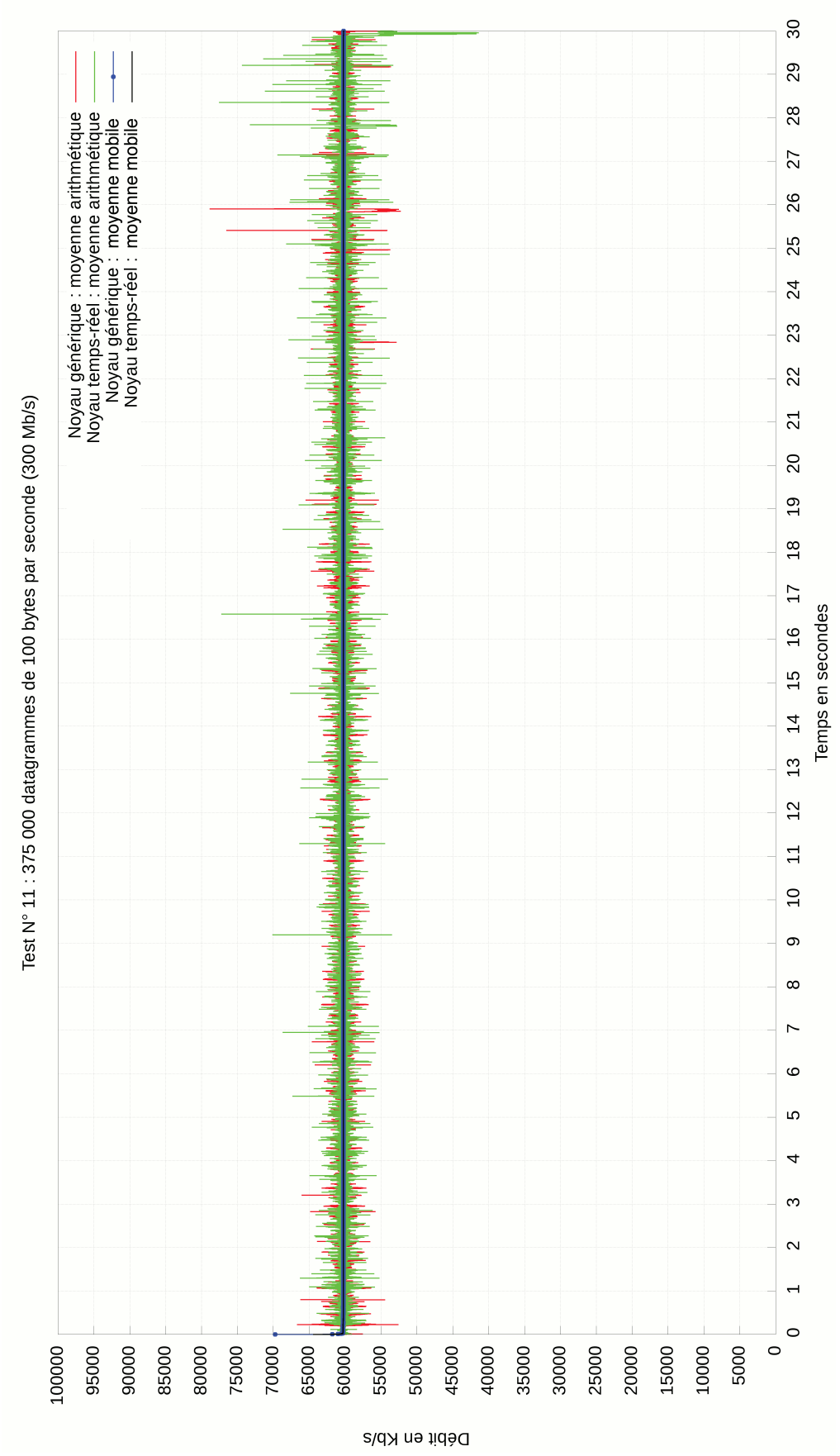

FIGURE F.11 – Analyse des débits – Test 11

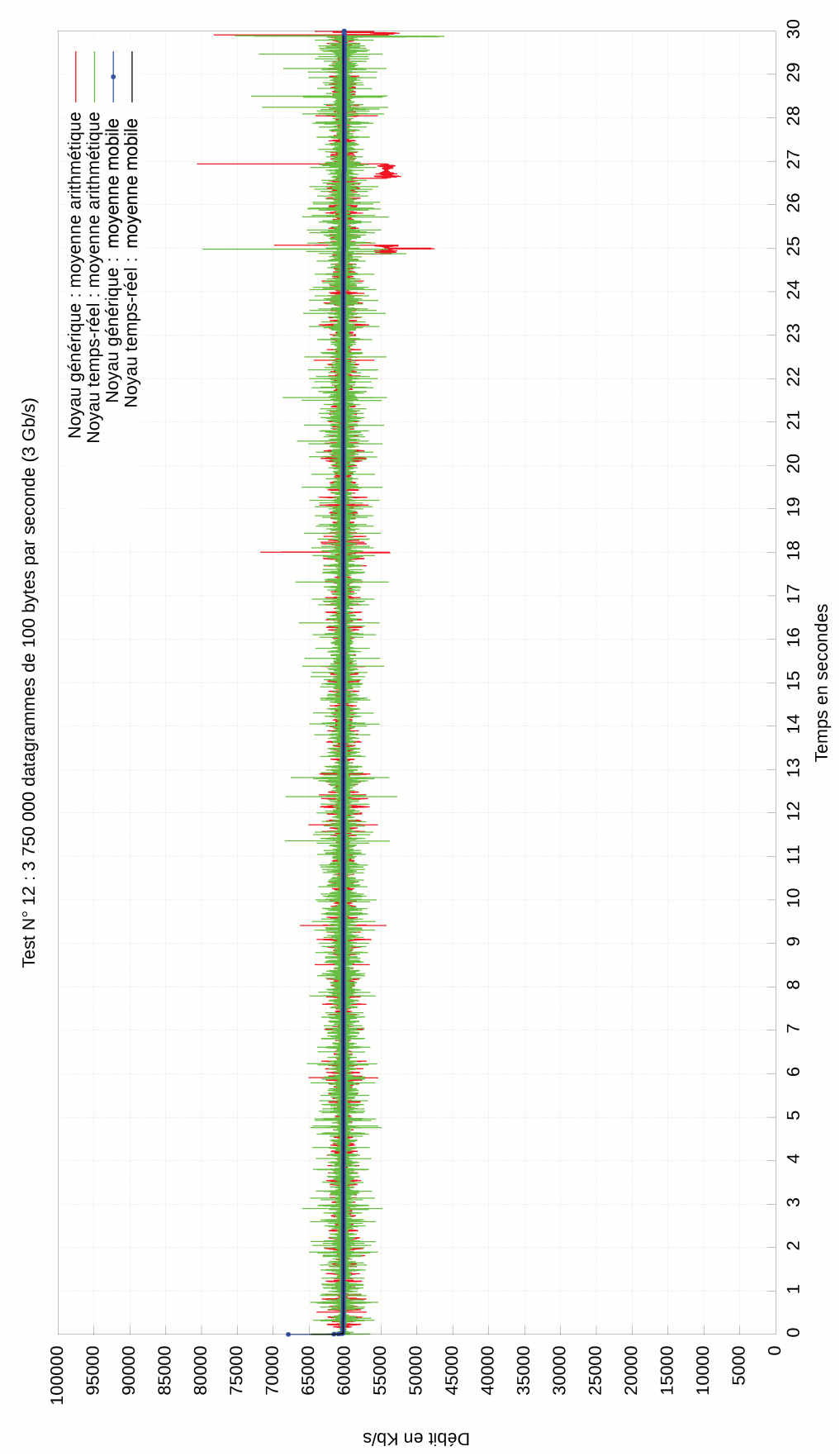

FIGURE F.12 – Analyse des débits – Test 12

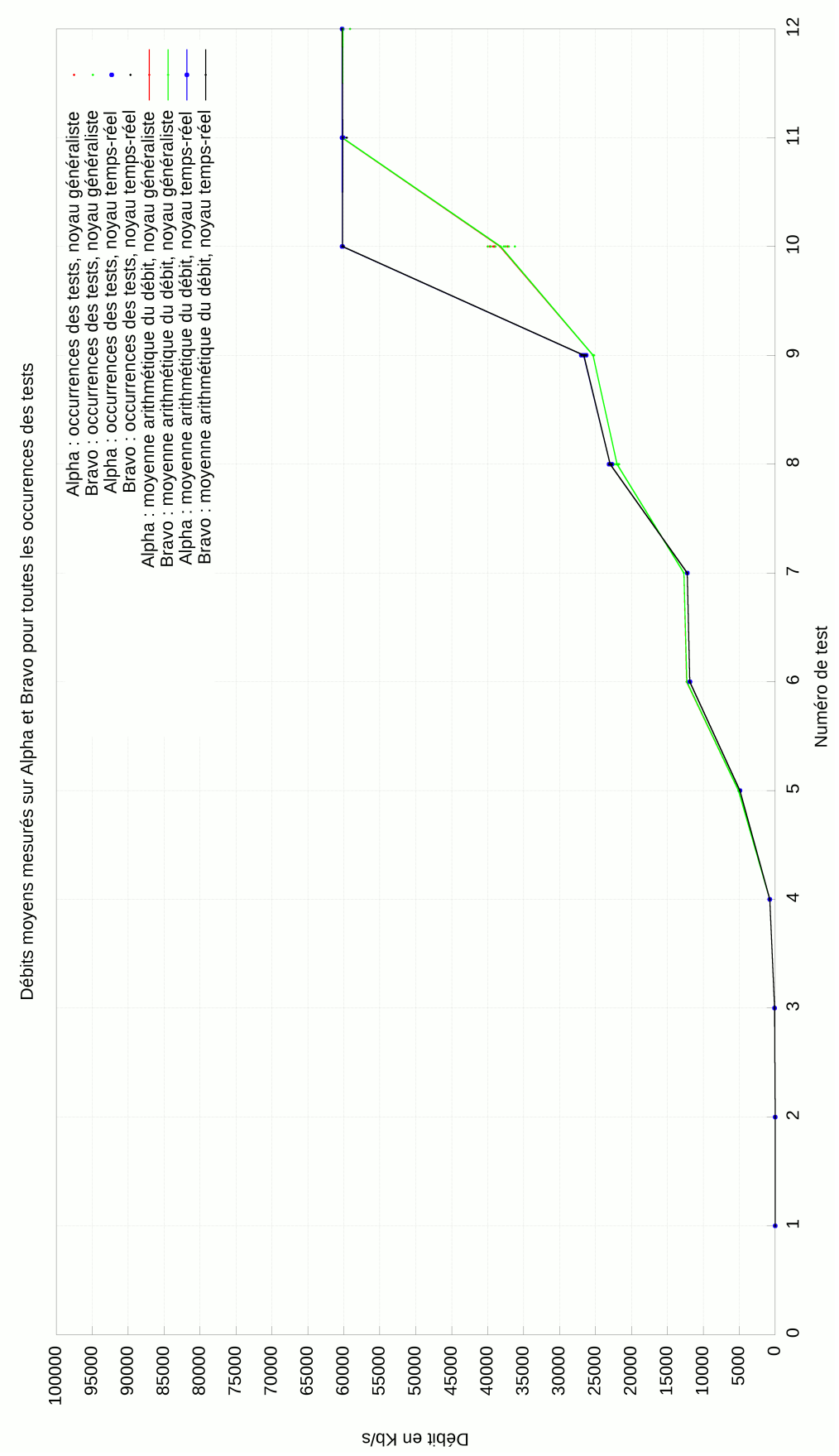

FIGURE F.13 – Évolution du débit moyen en fonction des tests

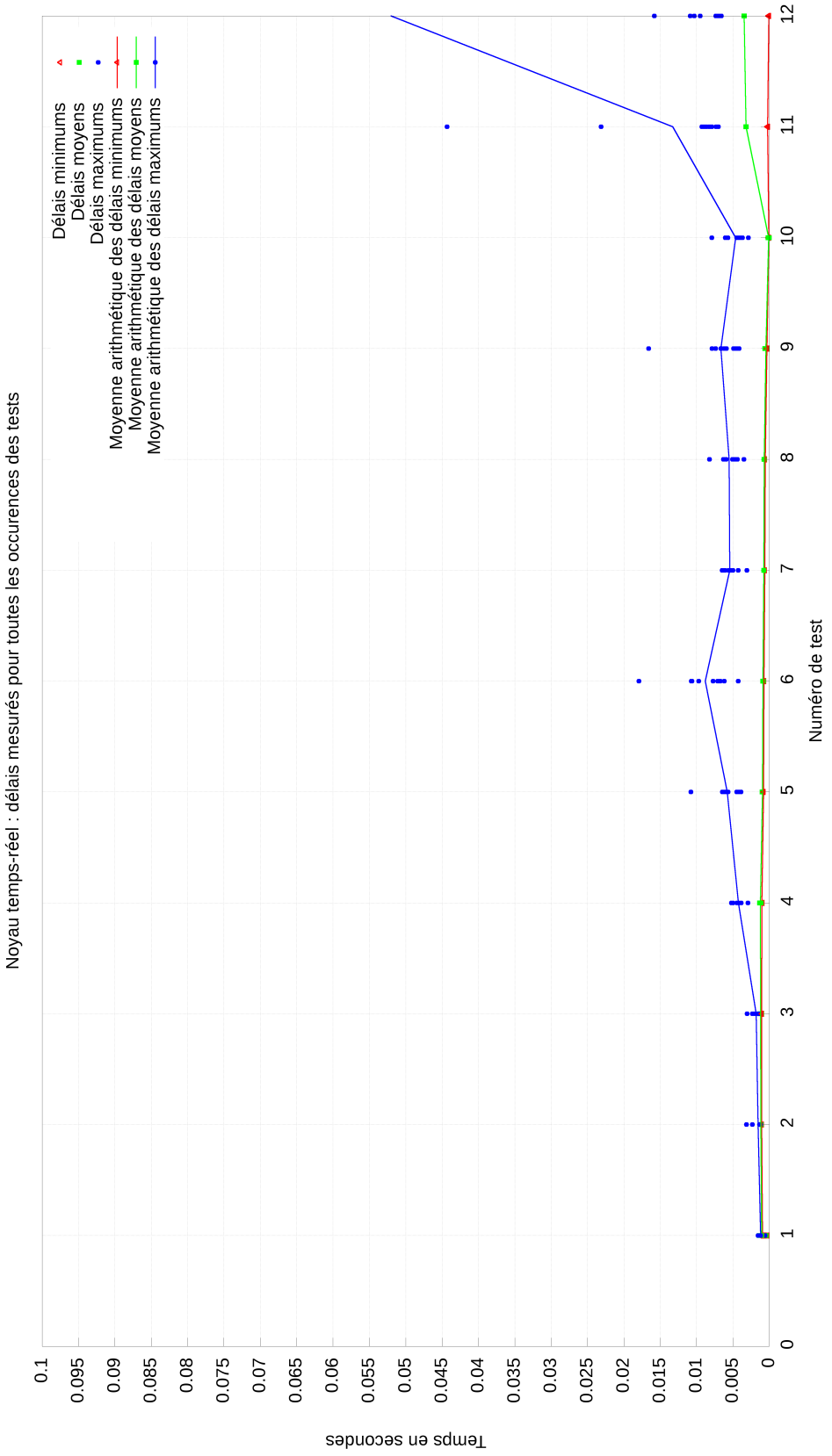

FIGURE F.14 – Analyse des délais – Noyau temps-réel

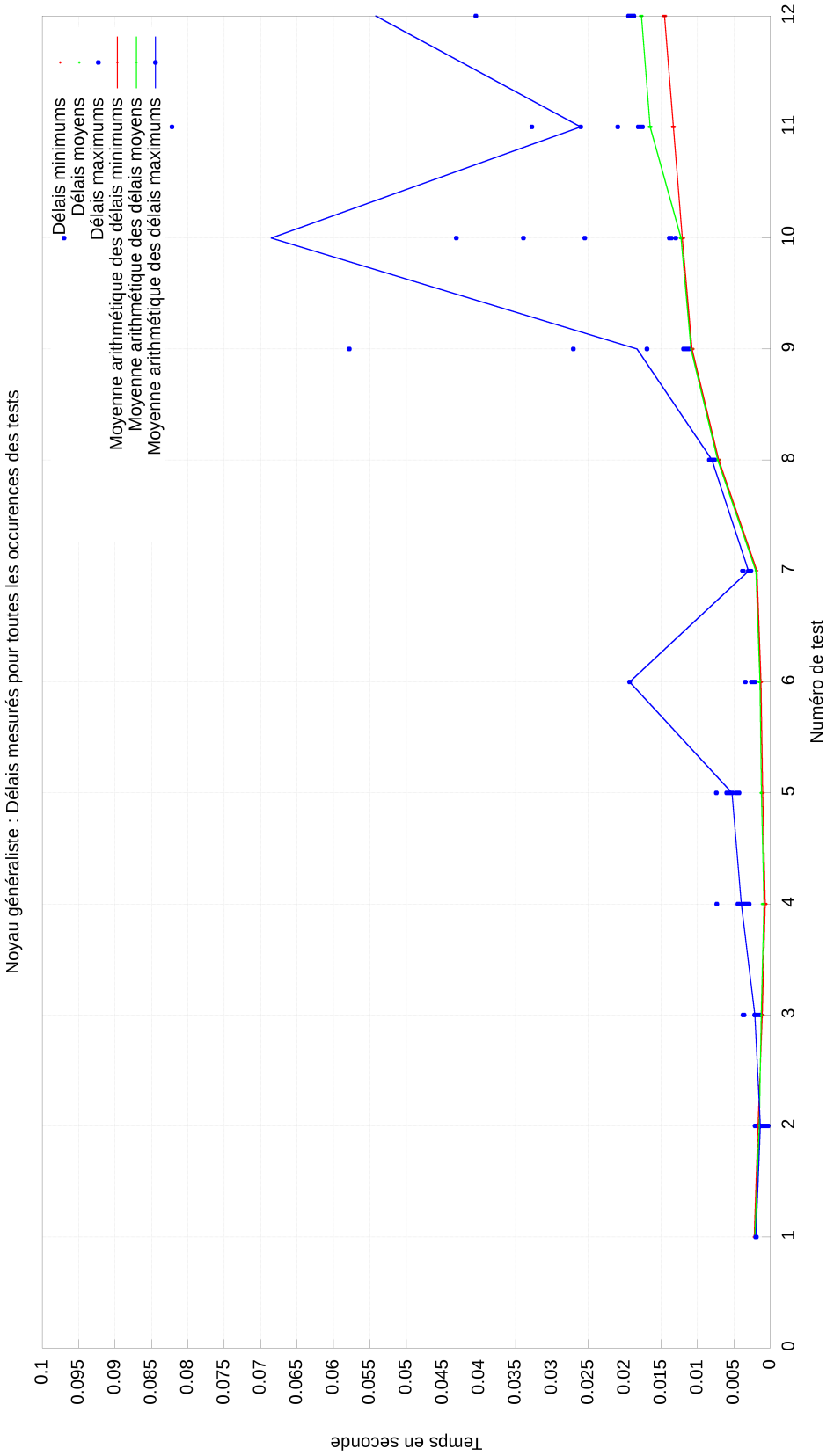

FIGURE F.15 – Analyse des délais – Noyau généraliste

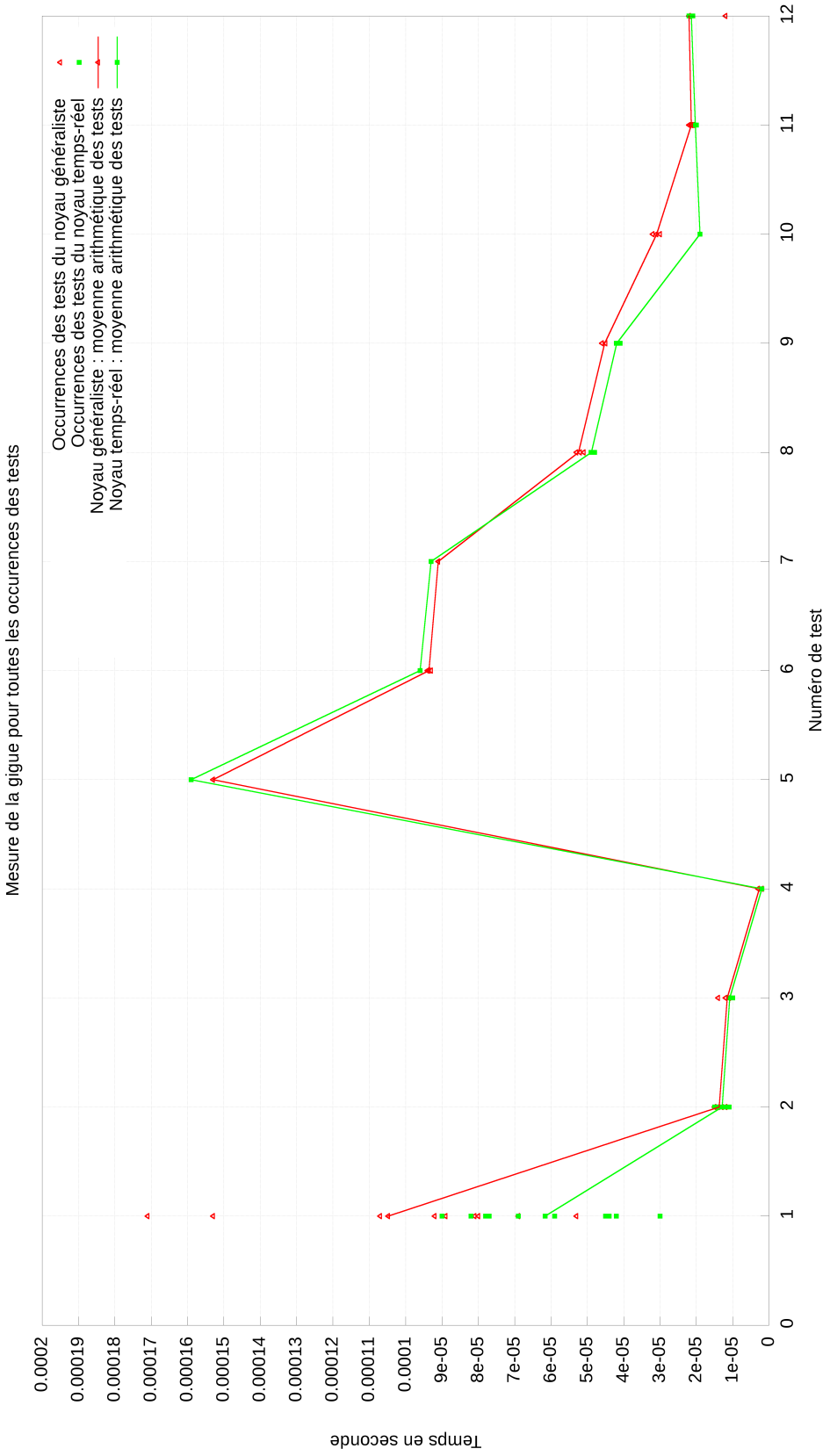

FIGURE F.16 – Analyse de la gigue – Noyau généraliste et temps-réel

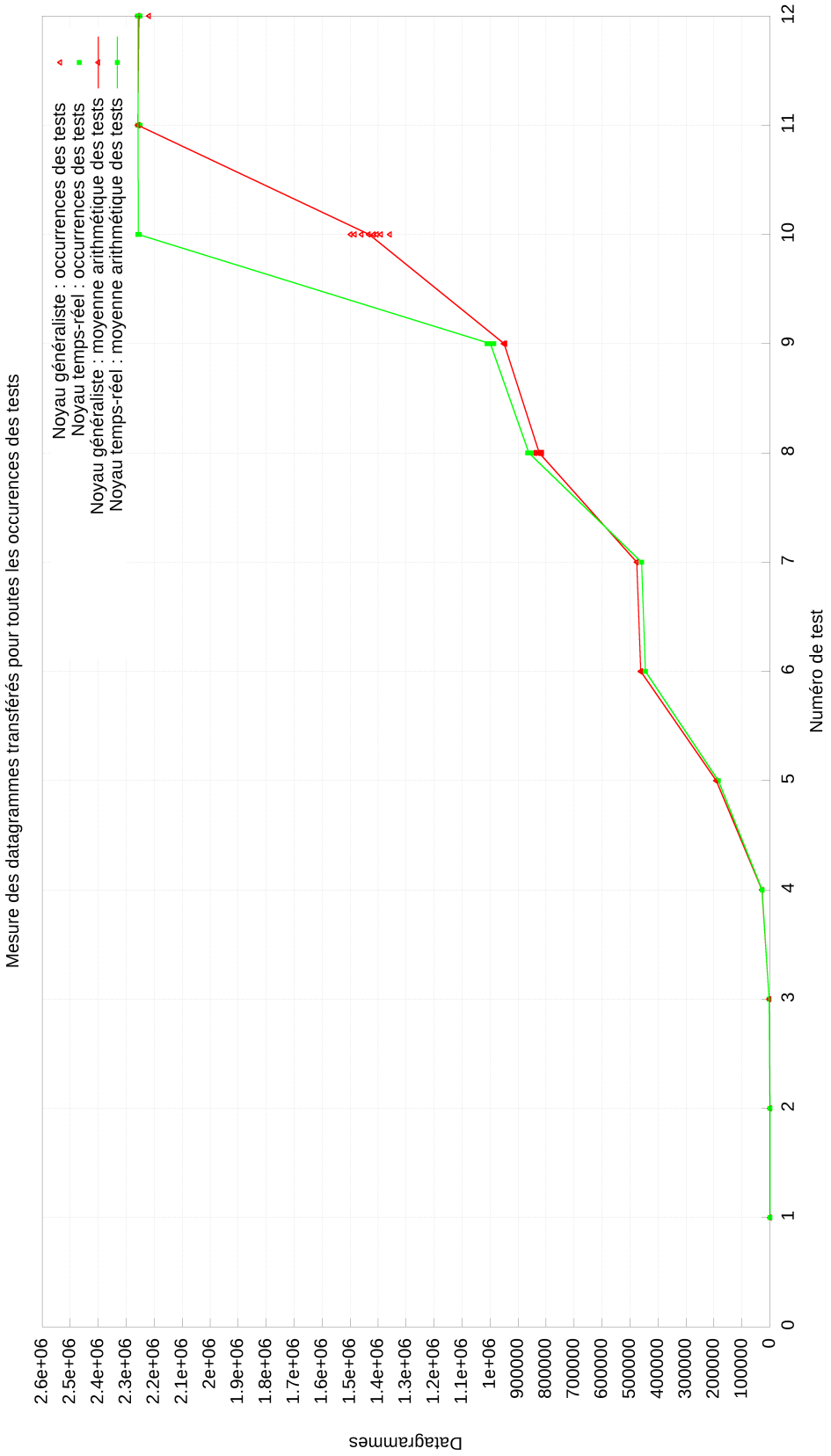

FIGURE F.17 – Totaux des datagrammes transmis – Noyau généraliste et temps-réel

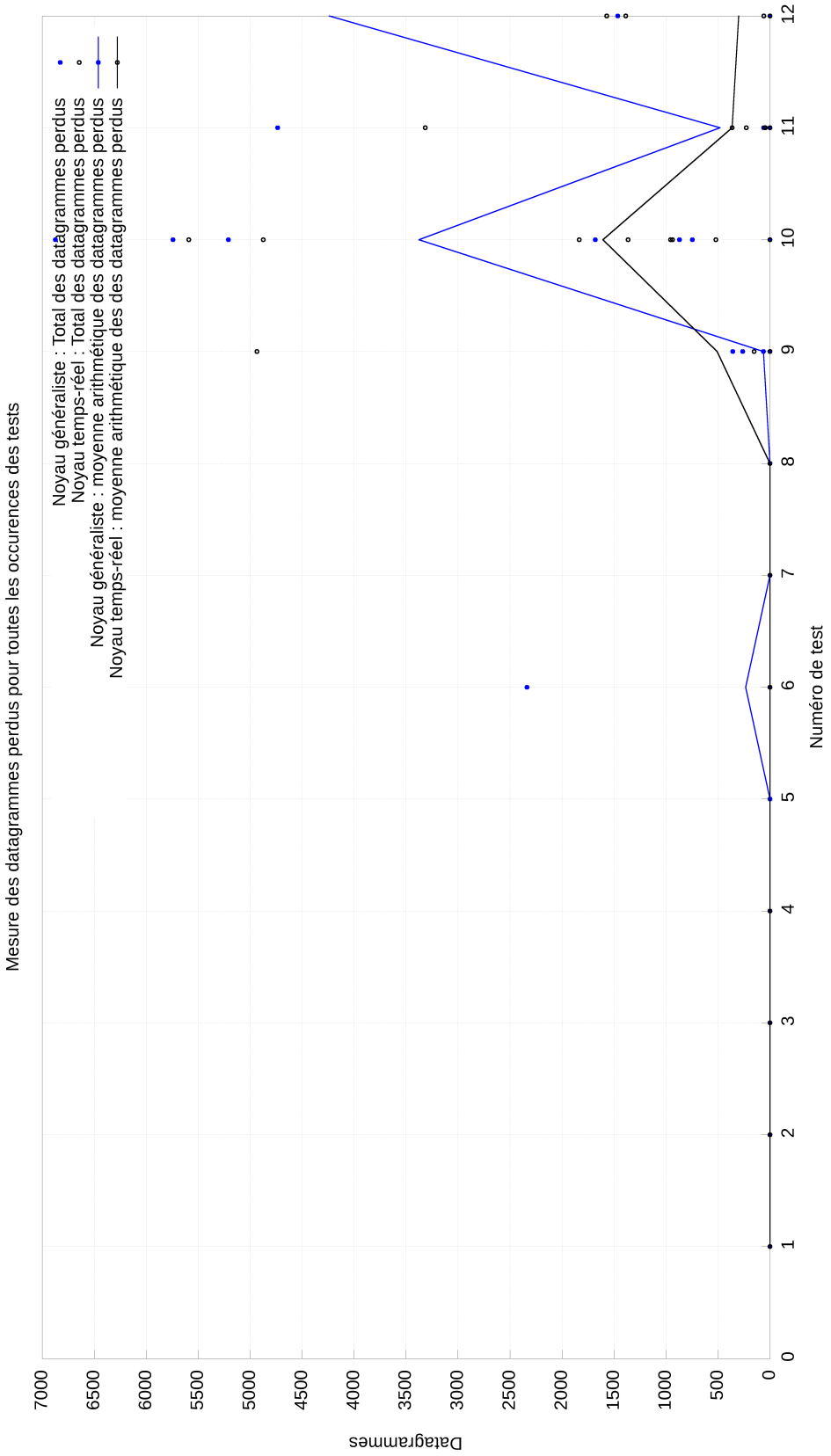

FIGURE F.18 – Totaux des datagrammes perdus – Noyau généraliste et temps-réel

Annexe G

Script d'automatisation des tests

```
# SCRIPT BASH DE GESTION DES TESTS
# -----------------------------------
# Parameters :
duration=30000
instance=10
outputPaths=("./t/" "./g/" "./g/s/" "./g/r/" "./t/s/" "./t/r/")
RecvIP=192.168.0.11
SendIP=192.168.0.10
payload=100
bitRateTab=(1 10 100 1000 10000 100000 125000 187500 250000 312500 375000 3750000)

# colors
VERT="\\033[1;32m"
NORMAL="\\033[0;39m"
ROUGE="\\033[1;31m"
ROSE="\\033[1;35m"
BLEU="\\033[1;34m"
BLANC="\\033[0;02m"
BLANCLAIR="\\033[1;08m"
JAUNE="\\033[1;33m"
CYAN="\\033[1;36m"

# FUNCTIONS :

initializeDirectories()
{
    echo -e "$VERT Checking directories..."
    compteur=0
    while [ $compteur -lt ${#outputPaths[@]} ]
    do
        if [ -d ${outputPaths[$compteur]} ]; then
            echo -e "$VERT--> Directory '${outputPaths[$compteur]}' already exists."
        else
            mkdir ${outputPaths[$compteur]}
            echo -e "$VERT--> Directory '${outputPaths[$compteur]}' created."
        fi
        let "compteur +=1"
    done
    if [ -d "./plots" ]; then
        echo -e "$VERT--> Directory './plots' already exists."
    else
        mkdir ./plots
        echo -e "$VERT--> Directory './plots' created."
    fi
    echo -e "$VERT""...done"
}

selectOutputPath() # in : outputPath not exists out : outputPath
{
    echo -e "$JAUNE""Select the output directory :"
    compteur=1
    while [ $compteur -lt $((${#outputPaths[@]}+1)) ]
    do
        echo -e "Enter value '$VERT $compteur $JAUNE' to select '$VERT${outputPaths[$(($compteur-1))]}$JAUNE' as output dir
        let "compteur += 1"
    done
    echo " "
    echo -e "Please enter your choice (no security over imput) :"
    read outputPath
    let "outputPath -= 1"
    echo -e "--> the output directory selected is : '${outputPaths[$outputPath]}'"
    echo -e "$JAUNE" "Please enter the name used to create the output directory"
    read outputDir
    outputPath=`echo "${outputPaths[$outputPath]}""$outputDir""/"`
    echo -e "--> The output directory is :" "$VERT""'$outputPath'""$JAUNE"
    mkdir $outputPath
    echo -e "$JAUNE""...done"
}
```

```
ITGDecGeneration()
{
    compteur=0
    echo -e "$ROUGE""Starting to deal with ITGDec..."
    while [ $compteur -lt $instance ]
    do
        echo "--> Generation of results $(($compteur+1)) / $instance"
        fileName=` echo "$compteur""_summary.d"`
        ITGDec $compteur > $fileName
        fileName=` echo "$compteur""_RawData.dat"`
        ITGDec $compteur -l $fileName
        fileName=` echo "$compteur""_delay.dat"`
        ITGDec $compteur -d 1 $fileName
        fileName=` echo "$compteur""_bitrate.dat"`
        ITGDec $compteur -b 1 $fileName
        fileName=` echo "$compteur""_packetloss.dat"`
        ITGDec $compteur -p 1 $fileName
        let "compteur +=1"
    done
    echo -e "$ROUGE""...done"
}

extractAndComputeAverages()
{
    echo -e "$ROUGE""Extracting data and computing arithmetical averages over all instances..."
    # create data files
    listeFichiers_d=("bitrate.d" "jitter.d" "delay_min.d" "delay_avg.d" "delay_max.d" "pkt_total.d" "pkt_loss.d" "bytes_rcv
    compteur=0
    while [ $compteur -lt ${#listeFichiers_d[@]} ]
    do
        touch ${listeFichiers_d[$compteur]}
        let "compteur += 1"
    done

    # happend data to files
    compteur=0
    while [ $compteur -lt $instance ]
    do
    cat "$compteur""_summary.d" | grep "Minimum delay" | sort -u | sed -e 's/[a-zA-Z= ]* //g'| sed -e 's/s//g' >> delay_min
    cat "$compteur""_summary.d" | grep "Maximum delay" | sort -u | sed -e 's/[a-zA-Z= ]* //g'| sed -e 's/s//g' >> delay_max
    cat "$compteur""_summary.d" | grep "Average delay" | sort -u | sed -e 's/[a-zA-Z= ]* //g'| sed -e 's/s//g' >> delay_avg
    cat "$compteur""_summary.d" | grep "Average jitter" | sort -u | sed -e 's/[a-zA-Z= ]* //g'| sed -e 's/s//g' >> jitter.d
    cat "$compteur""_summary.d" | grep "Bytes received" | sort -u | sed -e 's/[a-zA-Z= ]* //g' >> bytes_rcvd.d
    cat "$compteur""_summary.d" | grep "Total packets" | sort -u | sed -e 's/[a-zA-Z= ]* //g' >> pkt_total.d
    cat "$compteur""_summary.d" | grep "Average bitrate" | sort -u | sed -e 's/[a-zA-Z= ]* //g'| sed -e 's/Kbit\/s//g' >> b
    cat "$compteur""_summary.d" | grep "Packets dropped" | sort -u | sed -e 's/[a-zA-Z= ]* //g'| sed -e 's/(.\...%)//g' >>
    let "compteur += 1"
    done

    # create files to stock averages
    listeFichiers_sum=("a_bitrate.d" "a_jitter.d" "a_delay_min.d" "a_delay_avg.d" "a_delay_max.d" "a_pkt_total.d" "a_pkt_lo
    compteur=0
    while [ $compteur -lt ${#listeFichiers_sum[@]} ]
    do
        touch ${listeFichiers_sum[$compteur]}
        let "compteur += 1"
    done

    # compute averages
    compteur=0 #requested by output files
    for file in ${listeFichiers_d[@]};
    do
        # for each file
        avg=0
        for item in $(cut -d: -f1 $file);
        do
            # for each line in the current file
            avg=`echo "scale=7;$avg+$item" | bc -l`
        done
        avg=`echo "scale=7;$avg/$instance" | bc -l | sed -e 's/^\./0./g'`
        echo "$avg" >> ${listeFichiers_sum[$compteur]}
        let "compteur +=1 "
    done
    echo -e "$ROUGE""...done"
}

mova()
{
    echo -e "$ROUGE" "Starting to dealing with Mova"
    files=""
    for item in $(ls | grep "_bitrate.dat"); do
        files=`echo $files " " $item`
    done
    java -jar mova.jar -bv $files rollingAvg.d
    echo -e "$ROUGE""...done"
}

shiftingFiles()
{
    echo -e "$ROUGE" "Shifting files over directories..."
    mv ./*.d $outputPath
    mv ./*.dat $outputPath
    echo -e "$ROUGE""...done"
}
```

```
saveResults()
{
    # initialisation of the directries requierd to save datafiles
    initializeDirectories
    # select an output directroy
    selectOutputPath
    # use ITGDec to generate experiment results
    ITGDecGeneration
    # extract datas from the summary file generated before
    # put it in file with *.d extension and computing the averages of those values
    extractAndComputeAverages
    # compute the rolling variable with mova
    mova
    # shifting files over directories
    shiftingFiles


}

doTest() # in : test number
{
    if [ $1 -ne 0 ]; then
    echo -e "$ROSE Starting test number $1"
    compteur=0
    while [ $compteur -lt $instance ]
    do
        echo -e "--> Instance $(($compteur+1)) / $instance"
        ITGSend -T UDP -a $RecvIP -c $payload -C ${bitRateTab[$(($1-1))]} -t $duration -l $compteur -x $compteur
        let  "compteur += 1"
        sleep 10
    done
    echo -e "...End of test"
    sleep 5
    fi
}


# PROGRAM :
cpt=1
while [ $cpt -ne 0 ]
do
    echo -e "$BLEU" "Make a choice :"
    echo -e "------------------"
    compteur=1
    while [ $compteur -le ${#bitRateTab[@]} ]
    do
        echo -en "Enter value '$CYAN $compteur $BLEU' to do test number$CYAN$compteur$BLEU :"
        echo -e  " ${bitRateTab[$(($compteur-1))]} pkt/s during $duration millisecond(s)."
        let "compteur += 1"
    done
    echo ""
    echo -en "Enter value '$CYAN $((${#bitRateTab[@]}+1)) $BLEU' to run the 'Save results' utitility"
    echo -e " (about the last test)"
    echo -e "Enter value '$CYAN 0 $BLEU' to leave"
    echo " "
    echo -e "Please enter your choice (no security over imput) :"
    read cpt
    if [ $cpt -eq $((${#bitRateTab[@]}+1)) ]; then
        saveResults
    else
        doTest $cpt
    fi
    # clear
done
echo -e "Will now close the script..."
# clear
echo -e "$NORMAL"
exit 0
```

Table des figures

Liste des tableaux